Duckiesky Student Curriculum

Flight has fascinated humans for millenia.

The aim of this course is to empower people to build robots. Students will build, program, and fly an autonomous drone. This book covers everything needed to program an autonomous robot, including safety, networking, state estimation, controls, and high-level planning. Although the book focuses on an autonomous drone, we will provide a broad overview of modern robotics, including some topics relating to autonomous ground vehicles and robotic arms.

We will use the Duckiedrone to introduce concepts related to safety, control, state estimation, networking and communications, and mapping. Each student will build and program their own small quadcopter. After taking this course, students will be able to:

- Explain the space of designs for robotic communications, safety, state estimation, and control.
- Apply that knowledge to construct programs for communications, safety, state estimation, and control.
- Build, program, and operate an autonomous robot drone.

Contents

Section A
Introduction

Welcome to the DuckieSky Course! This section contains all of the background information needed to dive into autonomous flight. By the end, you will be equipped with the knowledge to consider where autonomous drones fit within the history of robotics, the ethics associated with such technologies, and the safety procedures required to use such technologies. You will also be equipped with the tools to edit this documentation should there be any confusions or mistakes in the text. We hope you enjoy the course, and if there is anything that is unclear or could use more details, we encourage you to leave a git “issue” to let us know (we’ll teach you how to do that in this section!) Enjoy!

Subsection A.1
Importance of Robotics

The lessons in this subsection introduce the course, provide a brief history of robotics, as well as discuss the complex ethical issues that arise as new technologies enter our societies. For example, should everyone be able to fly a drone with a camera wherever they want to? Or, are there ethical issues with this such as privacy, safety, and security?

Unit A.1.1
Intro to the Course

Module Overview

Robot: a machine (especially one programmable by a computer) designed to execute one or more tasks automatically and efficiently.

Quadcopter a more specific term used to refer to a drone that is flown with four motors.

Why is it essential to learn about this? How do robotics help our communities in society?

When we think of a robot, what definition could we reference?

Robotics matter because they can take autonomous actions in the real physical world, which can affect almost every aspect of human existence. It is an interdisciplinary research area part of Engineering and Science. It involves the design, construction, operation, and use of robots. The end goal is to create intelligent machines that outperform a human being's ability in a specific task or replace said human in a task too complex or far too dangerous. Making them near perfect multi use tools for industries and corporations of any kind.

Today robots impact the defense industry, manufacturing, the service industry, and more, with potential to positively transform work practices.

Useful Resources and References

1. Lexico.com Definition of Robot
2. Wikepidia What is a Robot
3. Wikepidia Info on Robotics
4. Sciencing.com Use of Robotics

Unit A.1.2
Development of Robotics

HISTORY

Evolution of Robotics Wired Episode 1

When we think of robots we tend to reference lots of pop culture films like; Star Wars with the famous duo of R2-D2 and C3-PO, The Iron Giant, Transformers, or the infamous T-1000 Terminators. But where does the word for these mechanical creations originate from?

Originally the word "robot" comes from the Czech language meaning slave. It was first used in the Czech playwright R.U.R (Rossum's Universal Robots) written by Karl Capek in 1921. Soon after the word "robotics" originated in the short story "Runabout" written by Russian-American author Isaac Asimov in 1942.

Robotic inventions have been around for a very long time. The concept of artificial intelligence manifested through mythology planted the seed for modern concepts of robots. For example, the giant Talos who served the gods of Greece was a bronze giant created by Hephaestus (god of invention and smithing) as an **autonomous** defense unit for the island of Crete. Another example are Golems part of Jewish folklore, which according to its definition are animated anthropomorphic beings created entirely from inanimate matter (**circuits or scrap metal**). Humanities ability to imagine artificial life before the time of intelligent or self moving machines is remarkable.

Inventions are documented as far as 3,000 B.C. in Ancient Egypt, following Ancient Greece, starting to take off in the 1700s, and began to be more prominent in the 19th century. Greek engineers and mathematicians made first designs of autonomy when inventing water clocks with self moving limbs and mechanical animals (The Pigeon) propelled by steam. In both America and Europe the Industrial Revolution (1760s - 1840s) introduced complex mechanics and the implementation of electricity making the powering of machines through compact motors possible. Although these inventions set foundations for the mechanical aspects, it was not until the 20th century that scientific efforts made great progress in Robotics leading up to the present day state of the art 21st century machines.

The first **digitally operated** and **programmable** robots were created by an inventor from Louisville, Kentucky named George C. Devol in 1954. Initially the programmable manipulator was patented under the name "Unimate" branded under "Universal Auto Animation", but was sold in the late 1960s to Joseph Engleberger after failing to be sold to the industry by its original creator. Engleberger happened to be a businessman/engineer and took the original design of the Unimate to redesign it into an industrial robot. Successfully marketing and mass producing the design gave Engleberger the title of **"Father of Robotics"**, making way into the designs for modern assembly line models.

THE IMPACT

Without a doubt the world we know today is reliant on much of the technology that was just concepts and schematics decades ago. The first use of modern robots evolved to aid humans in performing jobs too dirty or dangerous for workers that require higher levels of **precision**,**efficiency**, and **stength**. Commercial and Industrial robots are in use everywhere across the globe today. Manufacturing, assembly, and packaging are commonly assigned to serial robots which are composed of a series of joints and linkages. Parallel robots are another version of the serial model, with the difference being their smaller workspace and its arms being closer together. These two look like ceiling mounted spiders with super high speed loops in assembly lines for automotive components, computers, medical devices, or house appliances.

Robotics in medicine are used to facilitate surgeries and minimize invasive approaches when doing procedures on a patient. The most recent advances were showcased by the "da Vinci Surgical System"'. Being controlled by a surgeon from a console in the same room as the patient, the mechanical arms on this machine are so precise it is able to skin a grape without damaging the fruit. While there is lots of criticism on the use of these robots, it sure is an advancement in the medical field worth taking notice of. Another example aiding doctors in hospitals is the "Xenex Zapping Robot"'. Humans can not always sterilize a room 100% of germs and bacteria, but this robot combats the issue by pulsating full-spectrum UV rays that kill the harmful infectious microorganisms. A world of innovative robots is on the horizon as robots are constantly becoming more **sophisticated** and **responsive**.

While the Xenex Zapping Robots design is comparable to a droid from the Star Wars franchise, the following are orbiting and programmed outside of our atmosphere. In the past animals would be sent to space for research, but as technology improved that has now been delegated to robots. Records show that humans have physically only set foot on the moon and all scientific data on Mars, Titan, Jupiter, and Venus has been collected by robots. The first robot ever sent to space was the Sputnik 1 sent by the USSR in 1957. Its launch sparked the **"space race"** which brought to life better engineered space robots like the; Mariners, Vikings, and the Voyagers to be launched on space missions for close up photography for signs of life on these non terrestrial planets. The Vikings 1 and 2 arrived on Mars in 1976 disclosing both lander robots were powered by **radioisotopic thermonuclear generators** that enabled the transmission of data back to Earth. Close to present day achievements, SpaceX funded by Elon Musk launches version 2 of "C.I.M.O.N." (an artificial intelligent robot) to the International Space Station on December 5, 2019. It is equipped with cameras and voice commands that guide it, and it's even able to hold full conversations while relaying information to commanders on the ground. The acronym of its design stands for **C**rew **I**nteractive **M**obile compani**ON**, and is programmed for tasks equivalent to those of "Alexa" but for space.

1) What Does The Future Hold For Robotics?

With rise in computer industries, academia has advanced these inventions into the realm of A.I. (Artificial Intelligence) technology and while growth of this craft is expanding these robots have not made human workers absolete. But what does the future in robotics hold for the world? In 2013 the first safety standards for collaborative robotics was released, allowing for a broader use of robotics with regulations in other trades than the existing industries using robotic technology. Updates such as the Kalman filter, which is commonly used for guidance, navigation, and control in the field of robotics it applies motion planning and trajectory optimization. Putting that together with companies like Boston Dynamics who are recognized for their series of highly mobile robots and the the future of robotics seems to be getting closer and closer to what we may see on a scifi film, hoping the robot uprising is not part of the algorithm.

2) Vocabulary

1. Autonomous: denoting or performed by a device capable of operating without direct human control.
2. Programmable: able to be provided with coded instructions for the automatic performance of a task.
3. Radioisotopic: an unstable form of a chemical element that releases radiation as it breaks down and becomes more stable.
4. Thermonuclear: of, relating to, or employing transfomations in the nuclei of atoms of low atomic weight that require a very high temperature for their inception.
5. Digitally operated: operated remotely through a digital switch or remote.
6. Algorithm: a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

Useful Resources and References

Standford University History of Robotics

Standford University Ancient Lore of Robots

Wikepidia Definition of Golems

Thought Co. History of Robotics

Wikepidia History of Robotics

Acieta.com Industry Use

Future Market Magazine Transport Robots

Automation.com Future of Robot Industries

Bliley.com Space Robots

Washington Post Space X Robot Program

UNIT A.1.3
Intro to Ethics

1) What is Ethics?

The term ethics comes from the word “ethos”, which is Greek for “way of living”.

When there is a difficult situation, there are multiple possible solutions. Ethics consists of moral principles and values of a person or a group of people. It affects how we choose to live our lives, what we think is wrong and right in morals and situations, and what our responsibilities are.

Figure 3.1. Ethics (Image: scu.edu)

By considering ethics during decision making, we can make better decisions that would benefit individuals and society as a whole.

Specifically for ethical issues that are related to AI, they can be split into different categories.

The Ethical Implications of What AI Is

Bias and Fairness:

AI systems are being used more and more to simulate real life situations and tasks. In doing so, AI may be capable of amplifying human biases.

This may be the cause of biased data. However, another large focus of bias may be from the process of creating machine learning models. A single math equation is unable to perfectly represent all the data, so a math equation that gets as close as possible is used. In this process, the bits of data that originally had the largest representation would be altered the least by the final math equation. But others would get ignored by the model, which results in unfair representation (TowardsDataScience).

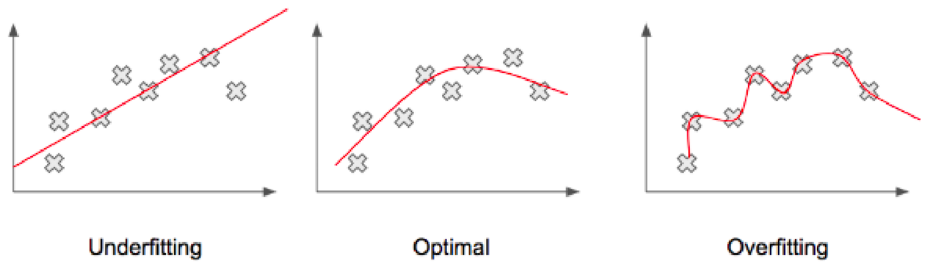


Figure 3.2. Optimal Line Vs Squiggly for Machine Learning Models (Image: pythonmachinelearning.pro)

Now what is "fairness"?

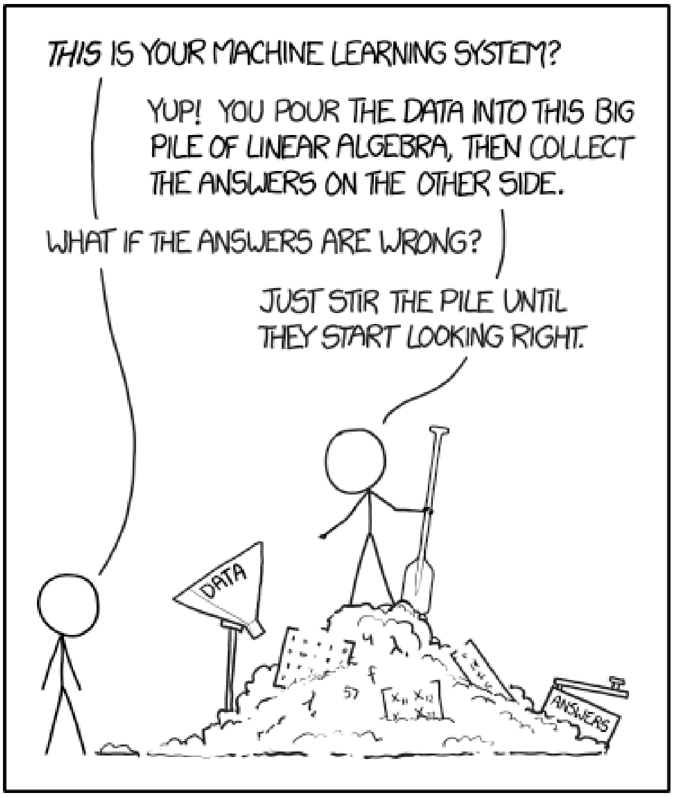

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Figure 3.3. But what is right and wrong? (Image: Machine Learning, XKCD)

There is not a specific definition for the term "fairness". It is a challenge that we are always striving towards: trying to create an autonomous system that is fair. To create a model that is fair, large understanding of many factors is required such as cultural, social, historical, political, legal, and ethical considerations. Many of these require trade-offs to be made. Should every group of people be given the same amount of benefits or should each group's given benefit be proportional to their situations? For different situations, it is always a difficult question and many have debates about what exactly is fair in a situation and who should be responsible for calling the final decision (Google).

Accountability and Remediability:

We know that there is always the chance that algorithms can create biased outcomes. Therefore, systems must be held accountable when this does occur. This can be done multiple ways, whether through getting humans to investigate the systems, assess impacts on users, or established policies.

And what happens when there is damage due to these systems and their biased outcomes? It would ideally be great if there are established policies or regulations that deal with this, but as of now, there are none. Currently, efforts have been taken by journalism and research teams to identify systems that are biased and push them to take action to resolve them (TowardsDataScience).

Transparency, Interpretability, and Explainability:

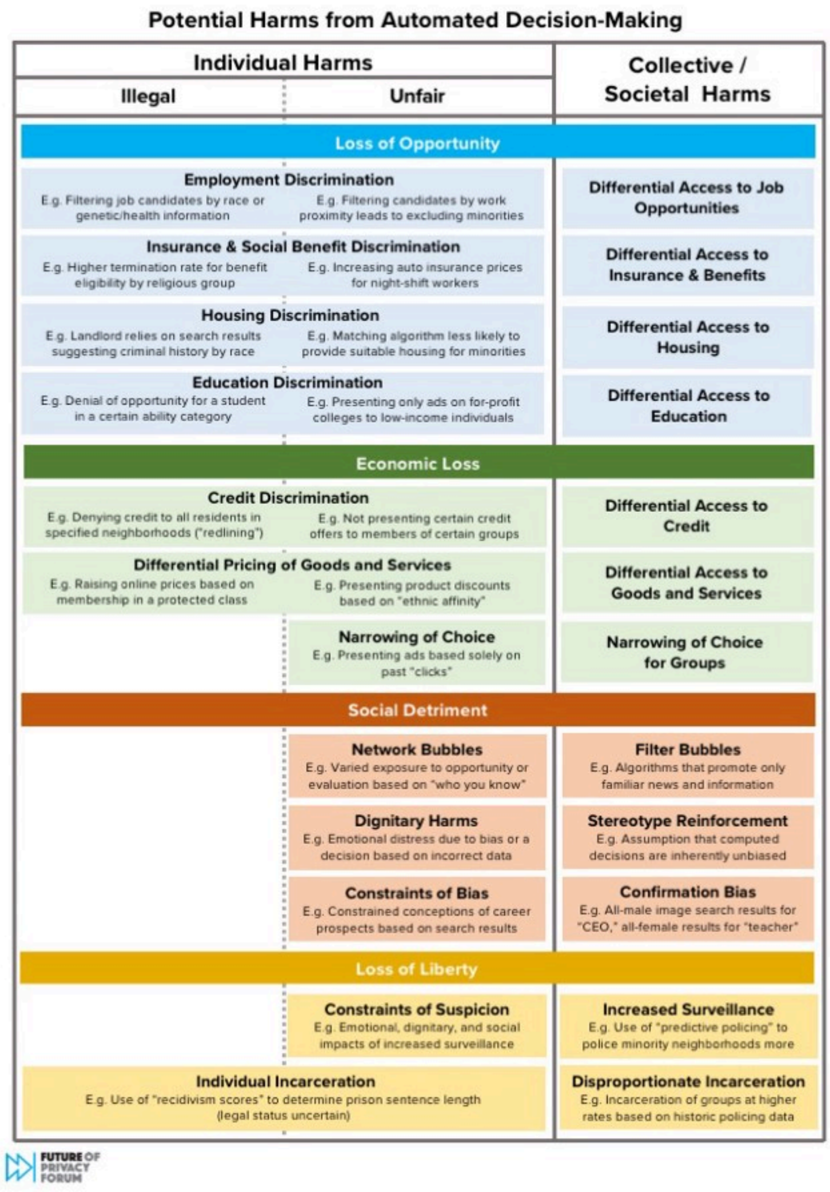
Many ethical issues arise because of lack of transparency, interpretability, and explainability.

Companies do not want to share their ideas, algorithms, and their models, which makes them reliant on data that is close sourced. This is due to multiple reasons: eliminate competition with other companies in related industries, and reduce hacking or malicious usage of their systems. However, this also prevents others from being able to identify possible sources of bias or inaccuracy in their systems.

Also, machine learning algorithms and models often require very complex math and multiple layers of it. This is because we want the math to be as close as possible to data patterns that are seen. Because of how complex it is, it is difficult for others to understand or explain on a detailed level how everything works (TowardsDataScience).

The ethical implications related to what AI does

Safety:



Potential Harms from Automated Decision-Making

Individual Harms		Collective / Societal Harms
Illegal	Unfair	
Loss of Opportunity		
Employment Discrimination E.g. Filtering job candidates by race or genetic/health information	E.g. Filtering candidates by work proximity leads to excluding minorities	Differential Access to Job Opportunities
Insurance & Social Benefit Discrimination E.g. Higher termination rate for benefit eligibility by religious group	E.g. Increasing auto insurance prices for night-shift workers	Differential Access to Insurance & Benefits
Housing Discrimination E.g. Landlord relies on search results suggesting criminal history by race	E.g. Matching algorithm less likely to provide suitable housing for minorities	Differential Access to Housing
Education Discrimination E.g. Denial of opportunity for a student in a certain ability category	E.g. Presenting only ads on for-profit colleges to low-income individuals	Differential Access to Education
Economic Loss		
Credit Discrimination E.g. Denying credit to all residents in specified neighborhoods ("redlining")	E.g. Not presenting certain credit offers to members of certain groups	Differential Access to Credit
Differential Pricing of Goods and Services E.g. Raising online prices based on membership in a protected class	E.g. Presenting product discounts based on "ethnic affinity"	Differential Access to Goods and Services
	Narrowing of Choice E.g. Presenting ads based solely on past "clicks"	Narrowing of Choice for Groups
Social Detriment		
	Network Bubbles E.g. Varied exposure to opportunity or evaluation based on "who you know"	**Filter Bubbles** E.g. Algorithms that promote only familiar news and information
	Dignitary Harms E.g. Emotional distress due to bias or a decision based on incorrect data	**Stereotype Reinforcement** E.g. Assumption that computed decisions are inherently unbiased
	Constraints of Bias E.g. Constrained conceptions of career prospects based on search results	**Confirmation Bias** E.g. All-male image search results for "CEO," all-female results for "teacher"
Loss of Liberty		
	Constraints of Suspicion E.g. Emotional, dignitary, and social impacts of increased surveillance	**Increased Surveillance** E.g. Use of "predictive policing" to police minority neighborhoods more
Individual Incarceration E.g. Use of "recidivism scores" to determine prison sentence length (legal status uncertain)		**Disproportionate Incarceration** E.g. Incarceration of groups at higher rates based on historic policing data

FUTURE OF PRIVACY FORUM

Figure 3.4. Examples of harms of AI systems and algorithmic decision making (Image: Future of Privacy Forum Report)

We want AI systems to be as safe as possible through decreasing the risk of bias, the risk of bodily harm, or display any behavior that can harm others.

AI systems are responsible for replicating functions of decision making skills of human minds. This involved having the ability to make decisions based on our ethics, intentions, and logical consequences that change depending on the situation and scenario.

Human-AI interaction:

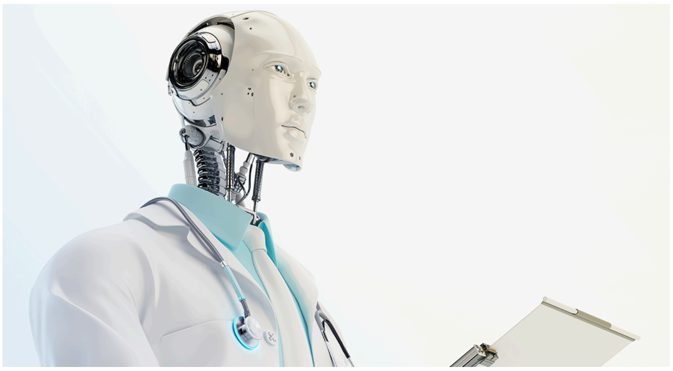


Figure 3.5. Automatic robots could be the future of nursing? (Image: ActiveAdvice)

While AI systems are made to replicate human minds or be able to do human tasks, there are many things that differentiate these systems from humans. However, as AI systems improve over time, these differences may become smaller and smaller, and can cause harm or provide benefits to people who use them.

Cyber-security and Malicious Use:

AI is becoming better and better at detecting, preventing hacks, and is now being used in many other systems as a cybersecurity measure. However it is still at a large risk of being hacked by others or used by others for malicious intent. This is due to the nature of AI, how it relies on input data and is constantly online for others to use 24/7, and many other reasons. Input data can be altered, others can constantly hack and alter key components of the algorithms through web packets or through the internet.

Privacy, Control, and Surveillance:

Figure 3.6. Technology can be a break down the walls of privacy (Image: Slane Cartoons Ltd)

AI systems can be quite easily converted for the purpose of surveillance. While it can be used for public good, naturally we would like to know exactly where and when our data will be used. But the catch here with AI systems is that others may be using our private data without our permission. AI systems learn from data provided during training, and they make predictions based on data input that they receive. The data may regard extremely sensitive or important data. There must be measures put in place so that the data can be used safely and efficiently by the system, while preventing the visibility of it from prying eyes.

2) Correctness and Uncertainty of Algorithms

By incorporating artificial intelligence (AI) into systems, they gain the potential to accomplish tasks that usually rely on the intelligence of humans. Systems can become autonomous, and do not have to rely on human control and decisions. An example of an autonomous system that is currently developing is autonomous cars.

AI systems utilize Deep Learning (DL) and Machine Learning (ML), which both rely on data matching and analysis algorithms, allowing the systems to replicate intelligence of human brains and enabling them to learn without human guidance.

There are benefits of using AI and autonomous systems and the use of algorithms in decision making processes.

Decisions can be made with more clearer and transparent criteria and choices will be less influenced by human emotions. The systems can learn from past actions and decisions that have been chosen and can analyze the consequences that resulted. If a result was non-favorable, then the system will remember and avoid picking the same choice for a similar future situation.

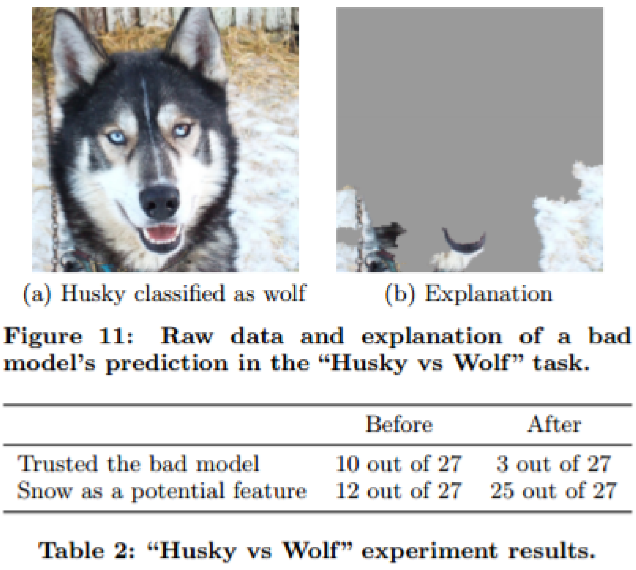
There are also disadvantages to autonomous systems or relying on algorithms to make decisions.

The algorithms may not represent all the factors that are related to a situation. There is also always chances that unexpected consequences will happen. There is no guarantee that a choice that had beneficial results in the past will still have the same result in the current scenario.

Also, the way that AI systems learn and their actions may become more unpredictable as they are given more complicated tasks that require more decision making skills (Yampolskiy).

Example: Husky Vs Wolf In Image Identification

University of Washington wanted to create an image classifier that can identify wolves from huskies correctly. The AI systems were fed images to learn from. However, some photos of huskies are incorrectly categorized by the system as wolves. It turns out that the system was learning from the images that wolves are often found in images that had snowy backgrounds. So the system turned out to be simplifying identifying if images had snow in the background (Medium).



(a) Husky classified as wolf (b) Explanation

Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

	Before	After
Trusted the bad model	10 out of 27	3 out of 27
Snow as a potential feature	12 out of 27	25 out of 27

Table 2: "Husky vs Wolf" experiment results.

Figure 3.7. Wolf vs husky algorithm detection experiment results (Image: Hacker Noon)

When there was a wolf in an image with no snow in the background, it would be categorized as a husky. If there was a husky with a snowy background, it would be categorized as a wolf (Ribeiro et al.).

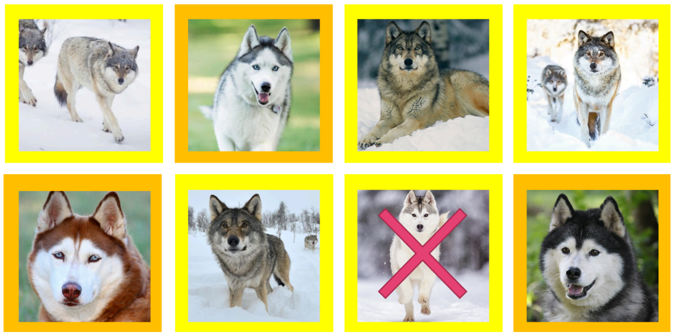


Figure 3.8. Input data of the wolf vs husky algorithm experiment (Image: Becoming Human: Artificial Intelligence Magazine)

This inaccuracy of the system is because of a data set that was "unfair" or did not have a sufficient variety of scenarios (Besse et al.).

Example: Artificial Neural Network Algorithm For Pneumonia Patient Risk

University of Pittsburgh created a study in the 1990s to use a system to predict which pneumonia patients were low risk and which were at high risk. The system initially caused a large amount of concern to doctors because they found out that pneumonia patients with asthma were classified as low risk by the system. A rule system was implemented into the system to help solve this issue. After examining data closely, researchers found that patients who had both pneumonia and asthma had a higher recovery rate. This is because when those patients were brought to the hospital, they were always considered to be at high risk, and immediately received proper treatment. However, the autonomous system simply believed that the presence of asthma results in being low risk, which is incorrect (Medium).

Example: The Boeing 737 MAX

There have been a number of accidents with the Boeing 737 MAX aircraft, which has resulted in the grounding of Boeing 737 MAX worldwide.

1. Lion Air Flight 610: domestic flight that took place on October 29, 2018. It crashed into the Java Sea shortly after takeoff. Resulted in deaths of all 189 passengers and crew.

2. Ethiopian Airlines Flight 302: international flight that took place on March 10, 2019. It crashed shortly after takeoff, and resulted in deaths of all 157 people on the flight.

There were several reasons that contributed to these fatal accidents:

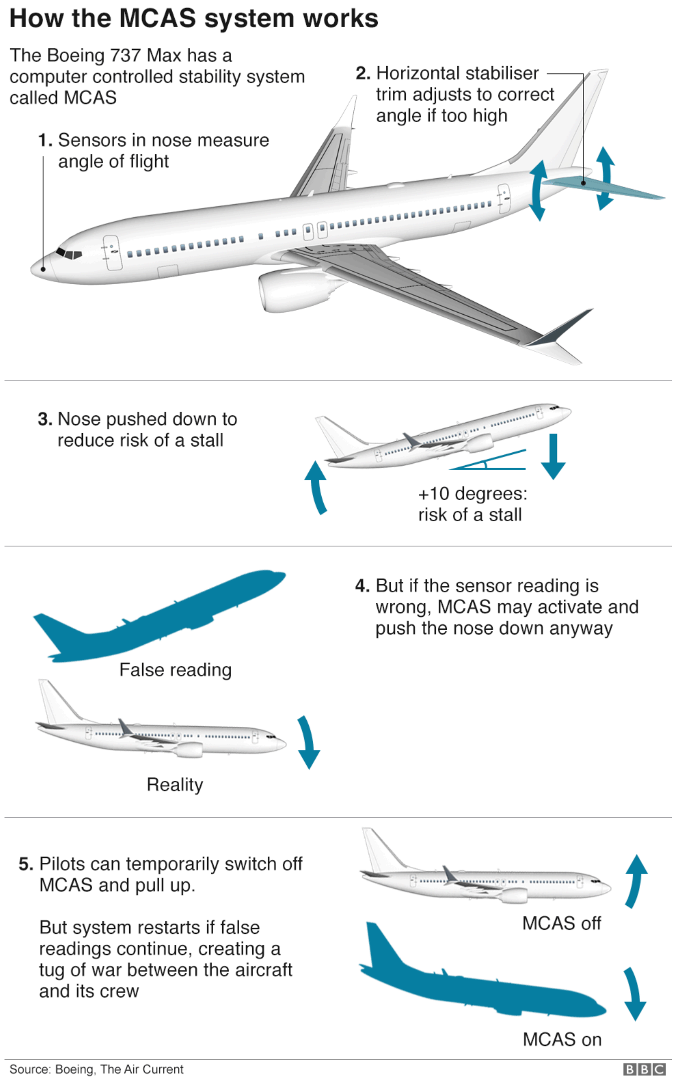

How the MCAS system works
The Boeing 737 Max has a computer controlled stability system called MCAS
1. Sensors in nose measure angle of flight
2. Horizontal stabiliser trim adjusts to correct angle if too high
3. Nose pushed down to reduce risk of a stall
+10 degrees: risk of a stall
4. But if the sensor reading is wrong, MCAS may activate and push the nose down anyway
False reading
Reality
5. Pilots can temporarily switch off MCAS and pull up.
But system restarts if false readings continue, creating a tug of war between the aircraft and its crew
MCAS off
MCAS on
Source: Boeing, The Air Current
BBC

Figure 3.9. MCAS flaws with the aircraft (Image: BBC)

The Boeing 737 Max 8 is different from the earlier Boeing 737 series. To allow for expanded seating capacity and better engines, Boeing 737 Max 8 had major design changes. The engines were moved forward and were raised. However, this made it more likely for the nose to pitch up while flying, so Maneuvering Characteristics Augmentation System or MCAS was developed to help correct the nose pitching problem by altering the control surface at the tail. The algorithm automatically detects whenever the nose pitches too high and corrects it (Seattle Times).

MCAS relies on only a single angle of attack sensor, instead of two. An angle of attack sensor helps warn pilots of a possibility of them losing control of the plane due to lack of lift (causing stall). Pilots are usually able to handle when the sensors are malfunctioning, however MCAS makes it a much larger problem. In both of the fatal accidents, MCAS was automatically switched on because of incorrect data from the single sensor (The Washington Post).

Pilots can temporarily switch MCAS off, however the system will restart and continue to work if the sensor continues to warn pilots of stalls. MCAS cannot ever been overridden by the pilots. The pilots lost control of the plane during the Ethiopian Airlines and Lion Air flights as the system was continuously fed inaccurate data from the sensor indicating that there are stalls, and they were unable to pitch up when needed, causing both flights to dive into the sea (The Verge).

Other flaws also contributed to the incident:

Insufficient testing:

Boeing and FAA agreed to not install safety features, which analysts say later that these features could have saved both the planes from crashing (The Washington Post).

Accountability:

Boeing did not provide the risk assessment about the MCAS until very late, a couple of months before the MAX was certified. FAA also based on findings by Boeing that were inaccurate (The Washington Post).

Lack of notice to pilots:

During their findings, Boeing calculated that an MCAS failure was also impossible. If it did happen, it is believed to be relatively low risk because according to the FAA, it is assumed that pilots can respond to unexpected situations within three seconds (Seattle Times).

Because of the low chances that were predicted, Boeing decided to not include the MCAS in the pilot manuals (Seattle Times).

3) Algorithmic Bias

✔ Exercise: Can try this game to learn more about bias in machine learning and algorithms.

Algorithmic bias can result from multiple sources.

1. The algorithm may be programmed by someone who is biased thus inheriting their biased views.

2. Since the systems often rely on pattern matching algorithms, an algorithm may act in a biased way because data that comes from biased sources.

3. The dataset given to the system to learn from could have biases that the developer is unaware of. Regarding the point made before regarding algorithmic uncertainty, from the data it receives, robots may unintentionally develop a biased and stereotypical way of thinking while trying to establish which factors it should prioritise during decision making.

4. Limitations of the mathematical model for machine learning of the data set.

5. Developers when testing their AI systems, do not test it with a large variety of data or do not take into consideration certain scenarios (which may come from a lack of diversity in the workplace).

Example: Amazon: Congress Matched to Criminals

Amazon created Rekognition, which is a facial recognition software. ACLU tested the software by matching Congress members, and the result was shocking: 28 members were matched with criminals. In addition, it was found that 40% of the inaccurate image matches were of people of color (ACLU NorCal).

This is risky to be actually implemented for law enforcement purposes, as it can make a police officer more biased before an initial encounter, or it can increase the chances of a person being questioned or searched, or can increase bias towards people of color (ACLU NorCal).

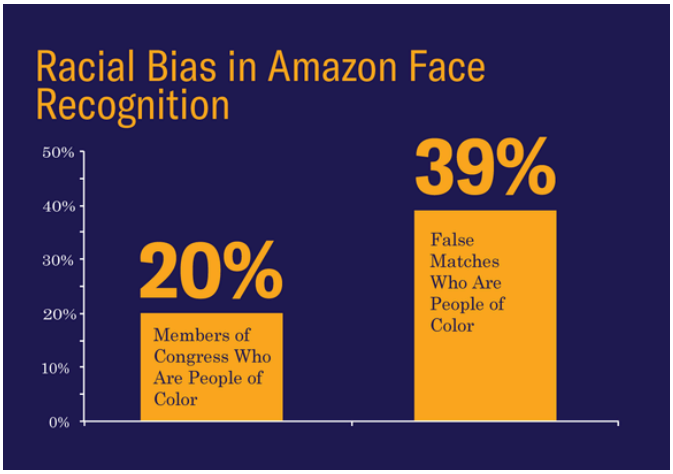


Figure 3.10. Congress Matched to Criminals Bias Rates (Image: American Civil Liberties Union)

Example: Autonomous Systems' Identification by Skin Tone

An autonomous soap dispenser by Technical Concepts was found to have trouble dispensing people for darker skin color. This is because the dispenser relied on IR sensors, which sense how much light is reflected back. Darker skin tones absorb more light than people with lighter skin tones (Reporter). This resulted in the soap dispenser not being able to work for people with darker skin. This design flaw was believed to be because of a lack of diversity in the workplace at Technical Concepts, who did not think to test their products on people with darker skin tones (Reporter).

According to a study done with autonomous systems by Georgia Institute of Technology, AI systems were more consistently accurately identifying people with lighter skin tones than darker. Their results show that detection of people with darker skins were less accurate by 5%. This can result in racial bias by the algorithm, and in the case of autonomous cars, people with darker skin would be more likely to be harmed or involved in an accident than those with lighter skin.

Pulling an example from earlier: Rekognition also displays the lack of accuracy with skin tone.

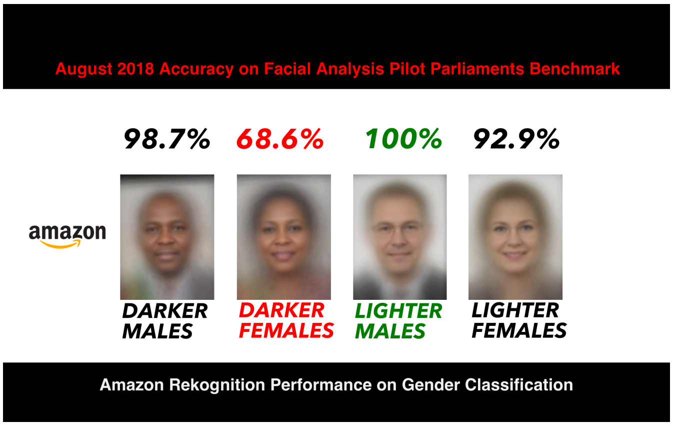


Figure 3.11. Rekognition's Accuracy Rates with Identification by Skin Tone (Image: Medium)

Example: MIT's Moral Machine

✔ Exercise: Try out some of the questions of the Moral Machine here website

In 2014, MIT created a series of questions and scenarios that involve autonomous cars and artificial intelligence, which is known as the Moral Machine. The Moral Machine asks people which choices autonomous cars should make when facing different variations of the trolley problem (Technology Review).

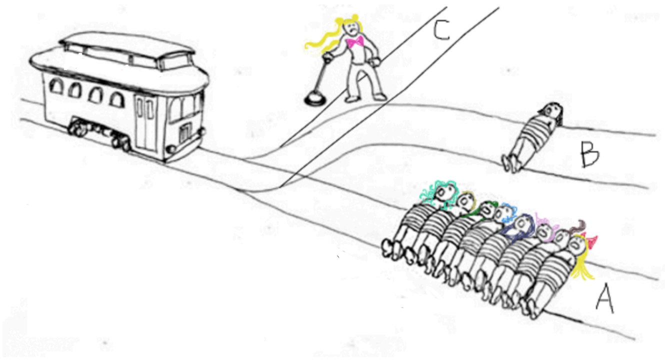


Figure 3.12. The Trolley Problem (Image: Medium)

The Moral Machine focuses on 9 different themes:

1. Humans or pets/animals?
2. Passengers or pedestrians?
3. More or fewer lives?
4. Women or men?
5. Young or old?
6. Healthy or those with health conditions?
7. People of those of higher or lower status?

8. Action or no action?

The results of the Moral Machine was closely related with culture and economics (Technology Review).

✔ Exercise: Read article to learn more about the experiment and findings.

4) Security and Systems Utilized in Society:

There are many systems in society that utilize autonomous systems that are important to society. There has been ongoing debate whether these systems should be close or open sourced.

Open source code means that the source code can be accessed by the public. Closed source code means that the source code cannot be accessed by others, or it remains classified, only seen by those who are authorized to. While closed source code may result in code being safer from prying eyes or hackers, it also prevents closer scrutiny for potential biases or problems in the algorithm by the public.

Example: Use of Biometric Data in Society

There are multiple forms of authentication and verification such as passwords, physical hardware keys, email, and biometric data.

There are many systems that use biometric data such as FaceID, fingerprinting, tracking down suspects, gaining access to restriction buildings, and access to important services.

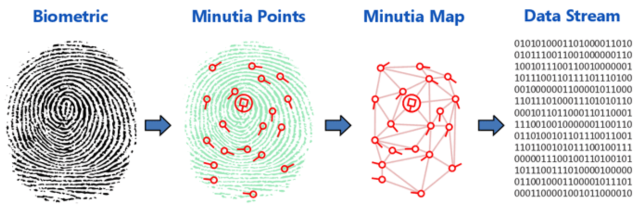


Figure 3.13. Process of using fingerprint data in systems (Image: Identity One)

There are many advantages of biometric data in society.

Many of the other verification methods have significant flaws or inconveniences:

Passwords can be guessed by hashing or brute force methods, only often requiring computation power. Passwords that are more complicated for computers to guess are often character sequences that are extremely difficult to remember, making it very inconvenient to use. Passwords are also very easy to change, especially for hackers who've already got past authentication.

Emails are relatively easy to hack and make it simpler to hack other personal information together, often done with the help of phishing emails that lead you to other malicious sites in disguise to ask you to authenticate into other services that you use.

Physical hardware keys are inconvenient due to the requirement of carrying them around and their ability to be easily lost or stolen.

Biometric data is relatively difficult to fake:

While biometric data is not completely fake proof, it does require effort such as taking very close up videos, or specifically recreating features from photos. Both of these require extreme close contact with a specific person or extremely good equipment.

_Biometric data is unique and difficult to change: _

Biometric data relies on unique features that you (or very few people) have, such as facial features or your voice. This makes it unique and secure. It is also difficult to change quickly by people with malicious intent, since biometric data may take some time to collect, process, and compress for usage in systems. It also makes it more convenient for users, since it does not require effort to memorize or carry around, your features are always present with you.

There are disadvantages of using biometric data in society.

Biometric data is dangerous to have lying around:

It is not safe to have compressed or complete biometric data present stored in systems or devices (for the verification process) where it could be vulnerable to cyberattacks. When retrieved, it can be easily used elsewhere or used for identity fraud.

Biometric data reading systems are not yet perfect:

Like what many of us have experienced with our smartphones, biometric data reading systems are not yet perfect. Usually it takes a couple of tries or different placements or orientations of your features to get it to work. And when it does get into a situation where you are unable to verify yourself, it is harder to get a more authentic verification of yourself.

Biometric data can still be faked:

As mentioned before, biometric data is not fake proof. If people with malicious intent do have extreme resources such as 3D printers or insanely good cameras, it is still possible to get a clear copy or reproduction of your features. There have been studies where 3D printed fingerprints are found able to bypass biometric authentication (Biometric Update).

Example: Autonomous Systems Used in Social Credit System Development in China

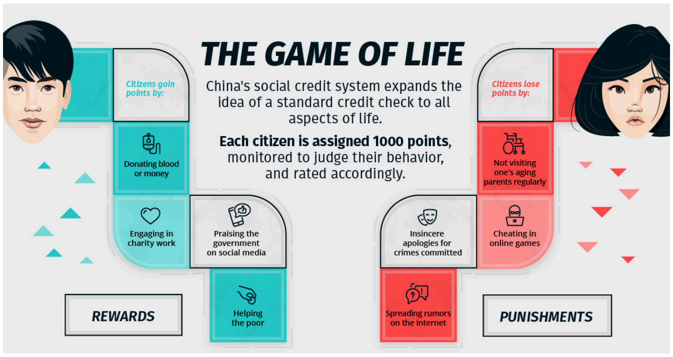

THE GAME OF LIFE
China's social credit system expands the idea of a standard credit check to all aspects of life.
Each citizen is assigned 1000 points, monitored to judge their behavior, and rated accordingly.
Citizens gain points by:
Donating blood or money
Engaging in charity work
Praising the government on social media
Helping the poor
REWARDS
Citizens lose points by:
Not visiting one's aging parents regularly
Cheating in online games
Insincere apologies for crimes committed
Spreading rumors on the internet
PUNISHMENTS

Figure 3.14. China's Social Credit System (Image: Visual Capitalist)

The Chinese government began experimenting with social credit scores in 2015, when it allowed private companies to assign credit scores to people. One of the companies, Sesame Credit, does this by analyzing many variables over five sets of data, most of which is actually from Alibaba's Alipay, which is used by over a billion people to make purchases, and contains much information. Sesame Credit analyzes both financial and social behaviors and assigns a score (Time). People who accomplish good deeds are awarded points and can receive rewards. Those who do bad deeds, such as smoking or spending too much time on video games, are deducted points. Those with an extremely low social credit score are placed on the "List of Untrustworthy Persons" and will be prohibited from certain activities such as the ability to use public transportation or to make large purchases.

This is being implemented with the help of AI and facial recognition.

Figure 3.15. Facial Recognition (Image: Biometric Update)

There have been many concerns with the use of China's social credit system, fears about leading to social inequality and alienation. There are also fears of preventing a free market economy. While the benefits include trying to reduce the amount of crimes or bad habits of people.

You can read more in this article

Example: Correctional Offender Management Profiling for Alternative Sanctions (COMPAS)

COMPAS is a software that is used by US courts to assign scores to predict the risk of a certain person committing another crime. It is an algorithm that utilizes an algorithm that considers answers to a questionnaire (The Atlantic).

In 2016, ProPublica has analyzed COMPAS and has found that COMPAS displays bias against African Americans.

You can read more about COMPAS here

Example: Unsecured/Exposed Robots Running on ROS and Internet:

A research team at Brown University discovered that they found almost 100 exposed systems that ran on ROS. "Up to 19 were considered to be fully operational robots". They found that they could access the cameras of the robots, and be able to give them commands for movement remotely (Brown University).

This shows how vulnerable systems can be when connected to the internet.

5) Militarization

There has been consideration of using autonomous systems for militarization. This could be for making military based decisions or using these systems to take action on made military decisions.

International Traffic in Arms Regulations (ITAR)

International Traffic in Arms Regulations (ITAR) specifically covers:

1. Covers military items or defense articles
2. Regulates goods and technology designed to kill or defend against death in a military setting
3. Includes space-related technology because of application to missile technology
4. Includes technical data related to defense articles and services
5. Involves strict regulatory licensing and does not address commercial or research objectives

(Source: Digital Guardian)

Example: Predator Drones Used by the United States

There have been several predator drones that have been used by the United States Air Force (USAF) and Central Intelligence Agency (CIA). Initially many are utilized for primarily reconaissance and spying, but many have been later equipped to be able to engage in warfare.

Autonomous Weapons Compared to Chemical Weapons

"In 1996, it was mandated that all stockpiles of lethal chemical agents must be destroyed. In 1997, the US ratified Chemical Weapons Convention Treaty and agreed to destroy any remaining stockpiles of chemical warfare agents no later than April 29, 2012" (CDC).

There has been much debate if autonomous weapons should be treated like chemical weapons? Should they also be prohibited from use and fully destroyed?

6) Medical, Healthcare, and Caregiver Robots

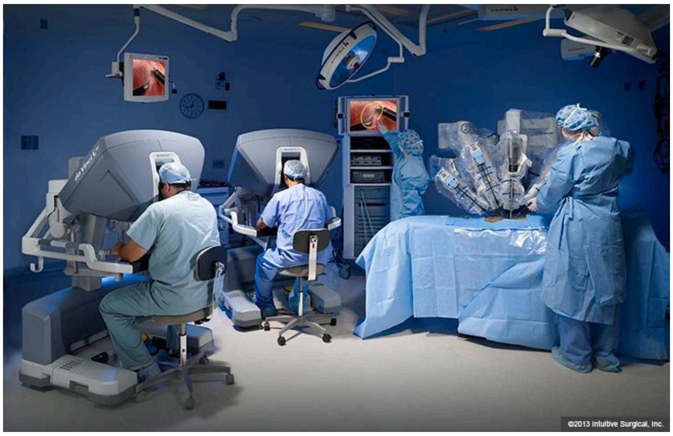


Figure 3.16. Robotic Surgery (Image: AARP)

Autonomous systems and AI systems have been also considered in the healthcare industry.

This includes the use of robots to help with identification, medical diagnosis, treatment, or surgery.

It also includes the use of robots to help keep elderly, disabled, young children, or patients company and to supervise them.

While there are many benefits such as being able to take care and monitor health of whoever they are responsible for and can provide them with company, there are several ethical concerns:

Privacy/Security:

Robots that spend a lot of time monitoring their patients or people that they are responsible for, may have the capability to be tapped into.

Especially in the medical field, these robots may contain snippets of important information such as health records, or video feeds of their patients.

Trust between robots and humans, and their interactions:

Humans have emotions while robots do not. Humans may have the capability to trust their robots too much, which can result in harm from the remote or manipulation of humans.

Example: The Emergency Exit Robot Study, Georgia Tech Howard

There was a study conducted by researchers at the Georgia Institute of Technology highlighted the potential risks of putting too much trust into robots during emergency situations.

They did an experiment that simulated an emergency situation. It was found that all participants of the experiment decided to follow the robot during the emergency, even if it led them through an noticeably incorrect path. Half of the participants have also seen the robot fail at navigating earlier before the specific experiment (Robinette et al.).

7) Availability/Accessibility/Uses

The cost of autonomous systems may be high depending on the purpose of the robot. The high prices of robots currently are barring many people from more complex robots.

Robots can be used to help guide people with accessibility issues.

Autonomous systems and advanced technology may also be used to help with emergency aid purposes. Drones can be used to help with potentially locating lost items or people, or helping transport emergency items quickly (water for forest fires ex.).

UN Guidelines for Emergency Uses of Drones

UAVs have potential in three areas: humanitarian, development, and peacekeeping operations (UN).

In 2013, the UN has launched the first UAV mission to help protect civilians in the Democratic Republic of Congo (UN News).

Read this article for more information: and article 2

Humanitarian Drone Guidelines

Figure 3.17. Drones can help with humanitarian efforts such as putting out fires (Image: WeRobotics)

According to Relief Web, the most promising uses of drones regarding humanitarian purposes include:

1. Mapping
2. Delivering lightweight essential items to remote or hard-to-access locations
3. Supporting damage assessments
4. Increasing situational awareness
5. Monitoring changes

Can read about more here

8) Future impact of AI on human jobs and responsibilities

With the development of AI, there has been a growing reliance on them as tools in our daily lives.

Ethical implications related to what AI can impact:

Automation, Job Loss, Labor Trends:

With the automation of many jobs, people may lose those jobs to robots. This may be because some tasks are able to be done more efficiently by robots. Because some jobs are more adaptable to robots than others, this may produce labor trends. There will be new jobs created that are more oriented towards maintaining robots or certain jobs will die out.

Impact to Democracy and Civil Rights:

AI may have a strong impact on democracy and civil rights. AI may be able to automatically detect certain messages or actions that are not permitted and can be used to prevent them. There have been related concerns to that and the Chinese Social Credit System, and fear that it may create a society focused around surveillance and conformity.

Human-Human or Human-Agent interaction:

As explored slightly in the medicine uses of robots, there will be much more interactions between human and non humans/robots in the future, which is different from interactions between humans only.

9) Useful Resources and Links if Interested

1. Seattle Times Article for more information about the MCAS system for the Boeing 737 MAX:
2. Verge Article about other flaws involved in the Boeing incidents
3. Washington Post Article on the lack of notice to FAA about Boeing MCAS system
4. Paper on Predictive Inequity in Object Detection
5. Moral Machine Test
6. Paper on the Moral Machine Experiment
7. Interactive moral machine
8. Article analyzing results from different countries
9. Paper about Fairness involved in Algorithms that undergo ML
10. Article with more detail about Pneumonia and Asthma Risk System and Wolf Vs Husky Identifier
11. Article about Rekognition and its failed Congress classifications

Subsection A.2
Interacting with our Curriculum

This module provides an overview of industry-standard programs like GitHub and Markdown and ultimately will instill the skills for you to directly propose changes and raise issues with the DuckieSky curriculum.

UNIT A.2.1

Git and Github

1.1. Creating a GitHub Account

- Here are the instructions to create a GitHub Account:
  - Go to GitHub's Home Page
  - Click Sign Up and complete the process for creating an account with your information
    - You should also verify your email address when an email from GitHub is sent to you

1.2. Git and GitHub Information

- What are Git and Github?
  - Git is a widely used version control system.
    - Version control systems are software programs that allow programmers and code-based project workers to manage the changes to their code-based projects over time with new versions.
  - GitHub is a Git repository hosting service, or an online datastructure that is a basis for storing and presenting these code projects.
    - It's kind of like Google Drive, but for software programmers!

> **Note:** An open-sourced project is a project where the code used to make a particular program or application is availible to everyone.

example

Checkout these big name companies that have open-sourced GitHub projects! You can actually look at the code that makes things run directly:

- Twitter
- Netflix
- Adobe

- What is the purpose of learning about GitHub?
  - It is an industry standard for most code-based projects.
  - This very textbook is hosted through GitHub!
    - We have worked really hard to make this curriculum for you, but we know there are bound to be mistakes or sections that can be improved. As a result, we would like you to be able to edit and improve this textbook; the more helpful you are in helping us improve this textbook, the better the course experience will be for you and for students learning this curriculum in the future!

    - The next lesson, Markdown and Contributions, will cover the programming language each individual page of the textbook is written in. You will learn today about the overall structure of projects hosted through GitHub (like our textbook) and how to propose changes to a GitHub repo.

1.3. Learning Git and Github

- Here is a worksheet on GitHub defintions that you can use while going through the tutorials to keep track of the terms or afterwards to assess the material that you have learned!
- Learn the Basics of Git and GitHub here!
  - First, follow this Introduction to GitHub Tutorial.
  - Next, follow this Forking Tutorial.
    - For this tutorial, you do not need to use GitHub Desktop or a text editor as of now, but you will want to read the section so you know what clones and commits are and how they are done!
  - Last, follow this Issue Tutorial up until the "Notifications, @mentions, and References" section.

UNIT A.2.2

Markdown and Contributions

2.1. Learning Markdown

- We're going to be learning Markdown and Markduck today, and at the end of the Unit, you are going to be able to directly submit some material to change our textbook!
    - Follow this tutorial to learn the basics of Markdown
        - Markdown is a text-to-HTML conversion tool for web writers.

2.2. Learning Markduck

- What is the difference between Markdown and Markduck?
    - Markduck is a Markdown dialect that is very similar to Markdown in terms of syntax. It is the language that the majority of the Duckiesky High School Textbook (including this document) was written in. Markduck has many characteristics that make it possible to create publication-worthy materials.
        - Syntax is a set of rules and characters that define interactions for a programming language. Individual syntaxes for programming languages can give various characters and code completely different meanings.
    - There are a couple key changes in the syntax between Markdown and Markduck, but, as a whole, almost all of the features you just learned in the Markdown tutorial remain the same!
        - In Markduck, an unordered list is created by putting a dash (-) instead of (*).
        - Sublists are created by using tabs instead of spaces in Markduck.
        - In Markduck, headers are used to define pages and separate sections. For instance, the "2.2 Learning Markduck" section title is created with a heading of level two (##) whereas the title of this page "Markdown and Contributions" was created with a heading of level one (#).
    - Read 1.5 Figures, 1.6 Subfigures, and 1.9 Comments.
    - Quickly skim over the section on special characters to be aware of some of the other features that Markduck supports which you can't directly use in Markdown!

2.3. Editing the DuckieSky Textbook

Great! It's now time to put your newly acquired Markduck skills and GitHub skills from last lesson to use.

- You have the knowledge to propose changes to our curriculum!

✔ Exercise: Submit a pull request to change the next page **Unit A.2.3 - Students: Leave your mark here!**; Click on the the pencil in the top right corner of the page to be brought to our GitHub repo. View a video demo of an example pull request below.

Be creative but you should have in your edit:

1. Add a new section for yourself with a level two heading (##) or higher.
    - Don't use a level one heading (#) because that will create a new page!
2. Tell us why you are interested in this course!
3. Bold and/or italize words or phrases.
4. Use an ordered or unordered list.
5. Use a special tag to do something cool!

Once you submit a pull request, hopefully one of our team members will be able to approve it as quickly as possible!

1) Helpful Materials to Propose Changes

- Here is a video demoing an example edit to the student page. This walks you through the process of submitting a pull request on our repo, which you can use to propose direct edits to any Markduck page in our textbook.
    - As you learned in the last Unit, pull requests can be used to directly propose changes to a repository. Use this to propose changes anywhere in our textbook that you think you are able to improve (like a typo or a word/sentence/section). Don't worry too much about making an inaccurate change because a member of our team will always look over it before it goes through; just try your best!
- Here is a video that demonstrates how to open and close a GitHub issue on our textbook repository.
    - You can use GitHub Issues to propose alterations that deal with the structure and the hierarchical content of the textbook or to suggest changes that you are not in the position to fix yourself.

Again, thank you for helping us and the DuckieSky community!

UNIT A.2.3

Students: Leave your mark here!

Use your newly acquired Markduck skills and Duckietown knowledge to submit a pull request to edit this page!

Be creative but you should have in your edit:

1. Add a new section for yourself with a level two heading (##) or higher.
   - Don't use a level one heading (#) because that will create a new page!
2. Tell us why you are interested in this course!
3. Bold and/or italize words or phrases.
4. Use an ordered or unordered list.
5. Use a special tag to do something cool!

Once you submit a pull request, hopefully one of our team members will be able to approve it as quickly as possible!

> Check before you continue
>
> Remember that you need to add an extra enter before any new items like section headers, lists, or special tags for them to be recognized by Markduck!

MAKE CHANGES BELOW THIS POINT

3.1. Dev's Section

I am interested in this course because:

- I **love** drones!
  - Specifically, drones with 4 propellers.
    a. More specifically, drones with cameras.
- I enjoy making *drone curriculum*

Warning: I do not particularly like troubleshooting issues with drones.

3.2. Mrs. Jones Section v3

> Note: I am interested in this course because:

- **It is a fun way to teach about robots**
  - **Specifically Drones**
    - STUDENTS BUILD OWN DRONE

3.3. George's Section

I am interested in this course because:

I want to learning about drones

1. I would like to underdtand the mechanics of building a drone.
2. More specifically assembling the parts.

[Robotic club] [Pre-engineering internship]

3.4. Lucas Furtado Section

- I am interested in this course because:
- I **always** wanted to build a drone.
    a. but specifically, drones with cameras.
    b. Drones with 4 propellers
- I think this will help me with my building and **Engenering** skills
- I *want to learn how to program a drone but also what parts makes a drone* ps: This is my first time building a drone

3.5. Luisangel Morales

I am intrested in this course becasue it's something new and intresting so I thought it would be fun and exciting to do.

1. I want to learn and understand about coding
2. I also want to know and understand the mechanics on buliding a drone
3. This will help me understand how to program a drone.
    - This will also help me learn how to use basic skills on building and engenering.

3.6. Serigne section

- I am interested in this course because: I wanna learn wanted to build a drone

3.7. Nahiomy's Section

I am interested in this cources: because I want learn about **robots and drones**. See how create a drones and the names of the pices.

3.8. *HILSON'S Reason*

1) *The reason I chose this class was because I wanted to try something new.*

3.9. Ivaldino's section

I'm interested isthis book because some resons. First, I really like to use drone and I love echnology. Second, I like to learn more about parts that make it fly. Also take some picture with the drone, and have fun.

3.10. Sylvia's Section

I am interested in this course because:

- I wanted to try something that was way out of my comfort zone
    - Building a drone would be a good learning experience
- I want to know what goes into programming and coding a drone
    - Including how long the process of building the drone takes
- I've always wanted to buy a drone, but being able to build one is much cooler
    - Learning about all of the parts of a drone should also be a fun experience

3.11. Rebecca's Section

Hi people

I wanted to be in this class for multiple reasons:

1. I'll learn the basic skills for building a drone
2. I'll learn how to code and program a drone
3. And I could potentially use the knowledge I gain from this class and use it in the outside world

3.12. Elijah's Section

I want to learn how to make things like drones and robots because they seam interesting to me.

Subsection A.3
Drone Operation

This subsection covers the hardware and software that is used on the DuckieSky drone to make autonomous flight possible. Last, but most importantly, the final lesson introduces drone safety using a case study and referencing official FAA rules.

Unit A.3.1
Sensors and Actuators

Important Vocabulary:

KNOWLEDGE AND ACTIVITY GRAPH

Sensors - parts on a robot that allow it to *sense*, or estimate, its own conditions or environment

Actuators - parts on a robot that use energy to *interact* with its environment

Controller - connects the input from the sensors to create an output from the actuators in order to accomplish a goal

Sensors in your drone::

1. **Infrared Sensor (IR)** - measures the distance to an object (or the ground) using infrared beams, then uses the cable to report it

Figure 1.1. IR + IR Sensor Cable

1. **Camera** - observes 2D images of the world to allow the drone to determine its planar position and speed

Figure 1.2. Camera (Pi Cam) + Flexible Flat Cable (FFC)

1. **Inertial Measurement Unit (IMU)** - sensor on your flight controller (FC) that allows the drone to tell how it is accelerating and rotating in all 3 dimensions

Figure 1.3. Flight Controller (FC) with IMU

Actuators in your drone::

1. **Motors** - actuators that spin at a variable RPM (revolutions per minute) depending on how much power it recieves (quantity = 4)

Figure 1.4. 2 CW and 2 CCW Motors

1. **Propellers** - device with blades attached to motors to turn rotational motion into thrust (quantity = 4)

Figure 1.5. 2 Clockwise and 2 Counterclockwise Blade Propellers (with Extras)

1. **LED** - actuator that lights up that you will be using in some experiments

Figure 1.6. 2 Clockwise and 2 Counterclockwise Blade Propellers (with Extras)

Controllers in your drone::

1. **Electronic Speed Controllers (ESCs)** - small computers that react extremely quickly to accomplish to simply keep the motors spinning at a particular speed by sending it a variable amount of power based on the input it recieves (quantity = 4)

Figure 1.7. 1 ESC for each Motor

1. **Flight Controller** - computer that connects the IMU sensor to the ESCs and motors to react quickly in order to fly the drone at a particular angle

Figure 1.8. Flight Controller

1. **Raspberry Pi** - more powerful computer that accomplishes a complicated goal, such as flying at a particular speed or to a particular position (it excecutes specific code loaded via an SD card)

Figure 1.9. Raspberry Pi Model B

Unit A.3.2
Safety

1) Case Study: The Midair Collision in 2009

There was a collision between a private airplane and a sightseeing helicopter over Manhattan. The result was 9 deaths.

NTSB is the National Trasportation Safety Board. NTSB reports provide detailed accounts of transportation accidents.

There were several causes that contributed to this according to the NTSB report:

1. The limitations of the see and avoid concept: 9 seconds before collision, the helicopter was not seen from the airplane. 5 seconds before collision, the helicopter was still very far from view. 1 second before collision, the helicopter was too close to be able to act sufficient. By the time the pilot of the airplane saw the helicopter, there was no time/not sufficient enough time to avoid the helicopter.
2. Teterboro Airport local controller's non pertinent telephone conversation (New York Times). The controller was distracted as he was having a phone call. He did not communicate with the pilot and was unable to warn the pilot of a potential crash.
3. Inadequate FAA procedures and regulations: There are inadequate FAA proceuures for transfer of communications between ATC facilities along that area of where the crash occured (New York Times). As well, the FAA regulations that not allow for sufficient vertical separation between aircraft flying in the region (Prof Tellex slides).

2) FAA rules

The Federal Aviation Administration (FAA) is a US governmental boday that is responsible for regulating aviation and unmanned aircraft. It regulates flights outdoors, airports, air traffic management, certification of people and aircraft, and protection of US assets. FAA rules do not apply to operations that take place indoors.

Students are considered to be recreational users.

Faculty and staff are considered to be non-recreational users.

Here are some of the important safety guidelines by the FAA:

1. Fly at or below 400 feet
2. Be aware of airspace requirements and restrictions
3. Stay away from surrounding obstacles
4. Keep your UAS within sight
5. Never fly near other aircraft, especially near airports
6. Never fly over groups of people
7. Never fly over stadiums or sports events
8. Never fly near emergency response efforts such as fires
9. Never fly under the influence of drugs or alcohol

3) Where to fly:

You may fly your drone indoors if you have enough space, as FAA rules do not apply to operations that take place indoors.

Before flying outside, please check the FAA's website for the Chart User's Guide to check which airspace you are located in.

You may also use the B4UFLY app, which is created by the FAA to help recreational flyers to figure out where they can safely fly and/or any restrictions in a location.

4) Possible Sources for Danger:

There are several possible sources of danger that can result from the drone:

1. applying force to your body
2. energy discharge from the body
3. parts or propellers dislodging from the drone
4. electric shorts and fires

5) Safe Environment

The Bystander Effect:

The more people that are present, the less likely someone will help a victim during a situation.

Be wary of this, make sure that if there is a dangerous situation, be cautious and aware, and take action to help those who need it.

Make sure that you have a safe environment to fly indoors.

Make sure you have equipment:

1. Safety glasses
2. Gloves
3. Walls
4. Distance
5. Net (not required)

6) Pre-flight Safety Checklist

Before you fly, you should make sure:

1. Do I have my safety goggles on?
2. Do I have a safe space to fly? Is there the possibility of the drone flying away or collisions? Make sure the surface you are flying over is not reflective, and is not uniform in details. Preferably a highly textured planar surface (ie: poster board with a bunch of scribbles and shapes) If outdoors: make sure that you are flying in a safe space that adheres to FAA rules, double check restriction and regulations of your space. Watch out for trees, flying duckies, people, etc. If indoors: make sure that you have an adequate clear area where you can fly a drone around and no obstacles that the drone/you will crash into.
3. Make sure the propellers, motors, and flight controller are correctly placed and not

loose. With the battery disconnected: make sure that the numbers of the propellers and the motors correspond. Also, the arrows on the propellers should be visible from the top of the drone, and the arrows should be going in the same direction as the arrows on the motors. The propellers should not be able to spin freely around the motor shaft. Make sure the propellers are tightened down so that they cannot spin freely and there is no gap between the propeller, the motor, and the motor nut. Make sure that the flight controller is not loose to ensure that it won't wiggle around or fall out of position during flight.

4. Make sure there are no dangling wires or wires that are in the way of the propellers. With the battery disconnected, spin the props with your finger and make sure there are no wires in the way.

5. Make sure that the flight controller USB is connected to the Pi.

6. Inspect the battery for any problems. Check battery level when plugged in. Make sure that the battery is not swelling, puffing, or smoking (when and when not plugged in or charging). When the drone is connected, you can check the battery level on the web interface. The battery level should generally be above 10V, if it is lower make sure it is charged.

7. With drone connected, make sure that you are connected to the correct drone. You can check this by running the blinkpowerled.sh script in pidrone_pkg

8. With drone connected, make sure there are no node errors. Go through each of the screens using ` n, where n is a number 0-5, and make sure there are no errors printed out. It is normal that there may be an error at the top of the screen that says something about not connecting to ROS master, but that is OK because it takes a bit for ROS to startup. Make sure there is other text underneath this error, and that text does not also include an error message.

9. With drone connected, make sure that the IR sensor is working. While looking at the web interface, move the drone up and down and make sure you see changes in the IR graph on the web interface

10. With drone connected, check the kill switch and ensure that it works. Press the space bar to arm the drone, and you will see the motors start turning. Press the space bar again to kill/disarm the drone, and you will see the motors stop.

7) First Flight:

The first time you fly the drone or start the drone, there may be some situations you may experience/be aware of:

- Drone flips: incorrect propellor orientations
- Collisions: no side/back sensors
- Battery mishaps: replace batteries or charge them
- Excessive heat production of the drone: shorts in soldering
- IR Sensor not working: check soldering and voltages
- Motors not responding: check calibration and soldering, run calibration script in terminal
- Haywire flight: check if flight controller is steady, make sure to run calibration

script in terminal

Useful Resources and References

The OSHA Technical Manual on Industrial Robots and Robot System Safety

SECTION B

Electronics

This section introduces the concept of electrical circuits, how they are created, and their basic characteristics including voltage, current, and resistance. This section also contains tutorials on soldering, which is used to create circuit connections on the drone. The knowledge and skills in this section will prepare you to start the drone build!

SUBSECTION B.1
Circuitry

This subsection explains basic circuit elements, how to create a circuit, and fundamental circuit characteristics including voltage, current, and resistance. This subsection also demonstrates how changing voltages and currents can create signals that carry information about a system.

Unit B.1.1
Simple Circuits

Links that cover info taught in class

Energy Conversions Info

diagram of the circuit

Conductors vs. Insulators

Useful Resources and References

Comparing Electricity to Water

Ohm’s Law Diagram

Video on Electricity - scienceworld.ca/resource/static-electricity/

Glossary

Unit B.1.2
Voltage, Current, Resistance

Links that cover info taught in class

How voltage, resistance, and current relate mathmatically using Ohm's law

How to measure voltage, resitance, and current with the multimeter

Difference between AC and DC

Possible Hw Sheet

Useful Resources and References

https://www.wikihow.com/Analyze-Resistive-Circuits-Using-Ohm%27s-Law

Glossary

Unit B.1.3
Signals and Connections

Analog signals

Information can be transmitted through electricity:

- Encoding in voltage (Seen in an IR sensor)
- Encoding in frequency (Used for WiFi)
- Noise, which can come from various sources, can distort or block the information transmittion done by the former

Digital signals

Our number system uses base-10 (10 symbols) to represent numbers, but there are many other ways to convey information!

There's no reason we have to use 10 symbols, you can get away with just 2 – binary! (or other numbers, eight: octal, sixteen: hexadecimal) Here's an ascii table that shows binary and decimal. ascii table

Your teacher may assign you to construct a flippydo - a useful paper device that can help you translate our decimal system into binary! flippydo instructions video explaination

Useful Resources

Analog vs Signals Video

Fun Binary Game!

Glossary

Subsection B.2
Soldering

The lessons in this subsection introduce soldering to create circuit connections, demonstrate the types of soldering techniques used to build the drone, and offer soldering exercises to hone your skills before starting on your drone.

UNIT B.2.1
Intro to Soldering

To put the drone together, wires have to be connected to each other, to sensors, and other electrical components. To do this, connections are formed by heating and cooling the mallieable metal known as solder. The soldering iron is the tool to heat the solder. Remeber, solder can get very hot and fling out. It is necessary to have eye protection, as well as to not breathe in soldering fumes. To make the soldering of two wires together easier, use helping hands or long-nose pliers. Tinning is what you do before you solder, which is coating a metal with solder to help it join and flow electricity better to another wire. Here's a video that shows how to properly tin a wire.

Vocabulary:

☀ Soldering is when two metals (solder and metal) are melted together. You use solder (show solder), to join wires.

☀ Tinning is what you do before you solder, which is coating a metal with solder to help it join and flow electricity better to another wire.

Important Safety Tips:

- The soldering iron is hot, only apply to needed components
- Solder can fling out and is necessary to have eye protection
- Do not breathe in soldering fumes
- The metals that the solder is next to is also hot, so use helping hands or long-nose pliers

Here's a step by step on how to tin a wire:

1. Twist the loose wire ends into a tight twist.
2. Hold the wire stably in place with helping hands or a plier.
3. Apply a solder dot onto the iron (to increase heat transfer from the iron to the wire).
4. Apply the iron onto the wire about to be tinned. Wait a few seconds for the wire to heat up.
5. Apply the solder onto the wire NOT THE IRON , and let the solder melt and flow into and on the wire.
6. Wipe away any excess solder with the iron being careful to not fling it too aggressively so to avoid burning someone/something.
7. Remove the iron and clean it.
8. Let the wire cool.

> Note: A pre tinned wire is a wire tinned at the factory. It looks shiny on the tip and the end cannot be frayed. For this courses' purposes, you should cut the pre-tinned end, and strip the wire casing, and continue self-tinning as seen above.

Useful Resources and References

1. Detailed instructions of each component and tinning
2. Soldering Tutorial for Beginners in 5 easy steps
3. How to Tin a Wire
4. Glossary

Unit B.2.2
Building Skill

To solder with flux, simply follow the steps of soldering from the previous lesson, except before tinning the wire, stick the soldering iron into the flux and apply it on the wires. This will make the solder flow easier, but is not always necessary. Here's a helpful video that demonstrates this.

Through-hole soldering is another form a soldering used on the Pi on the drone. This is simply when a wire is stuck through a previously made hole on a PCB. To complete through hole soldering, tin the wire, stick it through the pre-made hole, and solder the tip of the wire to the metal surrounding the hole. This video is a great demonsration on this technique.

Glossary

Unit B.2.3
Troubleshooting

Cold Joint Troubleshooting - watch from 0:00 to 2:00

Using A Multimeter

Glossary

Section C
Build: 1

In this first part of the build, you will create a circuit to provide power to the Raspberry Pi from the battery. You will make the circuit connections using the soldering skills that you've gained from the Soldering subsection. By the end of this build part, you will be able to power up your Pi and connect to it on your base station!

Build Part 1 Instructions can be found in the Operations Manual

Section D
Computing and Networking

After Build Part 1, you should be able to power up your Raspberry Pi and connect to it. This section will introduce you to two of the most important concepts of computing in robotics: networking and bash. Networking allows your computer to talk to the drone, and bash allows you to navigate through files and run programs on the Raspberry Pi using just commands that you type!

Subsection D.1
Using the Pi

This subsection includes an overview of Networking, SSH, and Bash. After completing this subsection, you will be able to connect to the Pis, navigate in Bash, and blink its LED using a command line interface.

Unit D.1.1
Networking

1.1. 7 Layers of Abstraction

- If not for networking, we would not be able to connect to and run our drones, so it's important for us to learn the concepts of Networking!
  - Learn about the 7 layers of abstraction at our edX lesson.
    - What is outlined here is the basis of what is used for our computers, mobile devices, and even with our drones.

1.2. Basestations

- What is a basestation?
  - For our purposes, a basestation is a laptop or desktop (ie. not a tablet) with the ability to connect to WiFi over a network and that has the ability to run/read python.
- How can we control our basestations?
  - A shell is a programming language that takes input and gives the input to the computer and operating system to analyze and perform the task that the input asks for.
  - A terminal is a program that allows the user to interact with the shell.
    - We will learn more about the terminal in the next lesson on Bash, which is the programming language that many terminals run in.
  - SSH (Secure Shell) is a method that allows a user to remotely log in from one computer/device to another. We utilized SSH to connect to our Pi in the past before we implemented the really handy text editor!
    - If you would like, you could attempt to follow our old build instructions to connect to the drone over SSH, but this may be a very self guided process (additionally it may not be compatible over Chromebook).

1.3. Networking with our Drone

- Connect to the Pi following the Build Part 1 Checkpoint Instructions from the Operations Manual
  - The text editor on this screen is Visual Studio Code (VSCode). This is a source editor that allows you to edit various files (like text and Markdown files). Use the editor to open and look at the files in the directory.
  - On the bottom of the screen is a terminal that runs in Bash. We will be learning about and directly utilizing this terminal in the next Unit.
    - If you can't find the terminal, follow the first 10 seconds of this video to open the terminal.

Unit D.1.2
Bash

2.1. What is Bash

- Connect to the Pi following the Build Part 1 Checkpoint Instructions from the Operations Manual.
- We will be learning about Bash commands and the terminal, as mentioned in the Networking Unit. It is important to learn how to utilize a terminal as it is the introduction to the inner processes of the operating system, and it will allow us to directly make changes to our drone!
  - We will be using Bash in the next Unit on Blinking an LED to directly see a change on our drone.
- Bash is the language that many shells are written in. Actually, our Pi's shell runs with Bash!
- Now, click on the terminal in the web editor for the Pi and prepare to input text.
  - Follow the first 10 seconds of this video if a terminal is not already open.

2.2. Learning Bash

✔ Exercise: Students should do the following steps with Bash commands to test out their knowledge of the terminal. They can do this through the online web editor when connected to the Pi:

- One of the features that a terminal can do is navigate and view the file system of a computer. For instance, folders you would typically find on your home Computer like "Desktop" and "Downloads" are directories. We are just now being introduced to this terminal, so we want to see the directories and files that are in the current location of the file system.
  - To do this, we can input (or enter) the Bash command **"ls"** that prints the files and directories in the terminal. Click on the terminal, type "ls", and hit enter.
    - The terminal will output (or print) all of the files and directories in the current location of the terminal. Remember, we are on the Pi's terminal so all of these files and directories are on your Pi.
- Folders can be within folders, so directories can be within directories! Your terminal is probably in a directory right now; we want to know where we are in our file system, so lets print the current directory name.
  - The **"pwd"** command prints the current directory: enter "pwd" into the terminal.
- Now, we know what directory we are in! Let's say we need to create a directory with a text file in it. There is a convention in Computer Science to print/use "Hello World" when being introduced to a Computer Science principle/language. So, let's name the directory "Hello" and the text file "World". The end goal of this mini-exercise is to print out the text file, which will contain the contents "Hello World".
  - The **"mkdir"** command allows you to make directories: enter "mkdir Hello".

- Check that the directory has been created by listing the current files/directories
  - Enter “ls” and there should be a new name that represents our directory in the list.
- We now want to enter into the “Hello” directory to make the “World” file.
  - The **“cd”** command allows us to traverse directories: enter “cd Hello”.
  - Enter “pwd” to see that we are in the new directory.
  - Enter “ls” to see the contents of the directory (which should be nothing because we just created it).
- We can create a text file called “World.txt” within the “Hello” directory at this point.
  - One of the functions of the Bash command **“touch”** is to create files: enter “touch World.txt”
  - Enter “ls” to confirm that the “World.txt” file was created.
  - The **“cat”** command prints text files into the terminal output: enter “cat World.txt”. There should be no output because we created an empty text file.
- We need to consider how we can input the contents “Hello World” into the World.txt file. To do this normally, we would just open World.txt in a text editor, type in “Hello World”, and save the file, but let’s say we only have access to this terminal and we don’t want to open any external applications.
  - The **“echo”** command will echo any input back to the output of the terminal: enter “echo I Love Drones!” just to check. You should get an output of “I Love Drones!”.
  - A carat (”>”) with a file name following it added to any Bash command will take the terminal output of that command and put it into the file.
  - We can use the carat to take the output of the echo command and place it in the World.txt file: enter “echo Hello World > World.txt”.
    - A cool feature of the carat is that it automatically creates a text file for you if one does not exist. In fact, we didn’t even have to use the “touch” command to create the World.txt file before the above “echo” command!
- Print the contents of the “World.txt” file.
  - Enter “cat World.txt”.
  - Congratulations! You have completed this conventional introductory “Hello World” step!
- Before continuing, use the VSCode navigator on the left side of the text editor to navigate to the Hello directory and open the World.txt file for viewing purposes (if it’s not showing up, you might have to hit the “Refresh Explorer” option).
  - We could have used this to input the text into the World.txt file, but you may not always have this handy web editor when dealing with a terminal.
- Delete the “World.txt” file.
  - The **“rm”** command deletes files: enter “rm World.txt”.
- Leave the “Hello” directory.
  - Using “..” as the argument for the “cd” command will allow you to leave a directory, enter “cd ..”.

    - Enter "pwd" to confirm that you have left the directory.
- Delete the now empty "Hello" directory.
    - The **"rmdir"** command will allow you to remove an empty directory: enter "rmdir Hello".
    - Check that the "Hello" directory has been deleted and is not on the list of files/ directories by entering "ls".
- Once you are ready to move on from this exercise, you might not want to have all of the terminal output above lingering around. The **"clear"** command removes all previous output: enter "clear". You can use this at any point to remove previous input/output from the terminal.

2.3. Exploring the Pi's Directories and Files

✔ Exercise: Students should explore the files and directories of their drone's Pi using the following commands: "ls", "cd", "pwd", and "cat". When finished, navigate back to the starting directory.

Warning: Don't delete or change any of the files or directories on the Pi already!

comment

You can use the up and down arrow keys on the keyboard in the terminal to see previously entered commands. You can use this if you want to reenter a command or reuse parts of a command.

2.4. Using Bash

✔ Exercise: Create a folder that is called "Actuators", create three text files (name them "1.txt", "2.txt", and "3.txt"), and insert the name of a different Actuator on our drone in each of the three files.

Check before you continue

Make sure to delete the "Actuators" directory.

- This time, try to delete the directory without deleting the contents of the directory first.
    - You should get the following error: "rmdir: Actuators/: Directory not empty".
    - Instead, use the command "rm -r Actuators".
        - The "-r" is telling the rm command to delete recursively, which means it will delete all of the files and directories within before deleting the Actuators directory. This is outside the scope of this class, but if you are interested, you can read more about recursive functions here.

- Once you are finished, the **"exit"** command will stop the terminal: enter "exit".

UNIT D.1.3
Blinking an LED

3.1. Step 1: Blinking the LED in the REPL

There are two types of signals that an LED can provide: **High** and **Low** signals. To turn on the LED, we need to provide **High** signal on the pin we choose, while to turn off the LED we will give a **Low** signal.

After we power up the Pi, navigate to the code editor, and open up the terminal, we can type in this line and press **Enter**:

```
gpio -g mode 6 out
```

In this line, the -g option is the option to define the **GPIO BCM** pin as explained earlier. We can use this page to figure out the numbering. In this case, our LED is connected to **Pin 6** on the Pi, so we designate the corresponding **GPIO6** to run the **Output** mode, which means that it can provide **High** or **low** signal.

Next, we write the signal on GPIO6 to be High by running this command:

```
gpio -g write 6 1
```

At this point, you should see the LED turned on. Similarly, the following command will turn it off by writing the Low signal:

```
gpio -g write 6 0
```

Lastly, we can use the gpio blink command to blink the LED:

```
gpio -g blink 6
```

In the above command, we command pin 6 for a blink, turn on 1 second, turn off 1 second, and so on. Press **CTRL+C** to stop the command.

3.2. Step 2: Blinking the LED using a bash script

An alternative way to blink the LED is to use the gpio commands in bash. Before using them, we need to know the path to the gpio command by using the **which** command in the terminal:

```
which gpio
```

You should get an output of **/usr/bin/gpio** path.

Now we are ready to make the bash script. The first step is to create a file such as **blink.sh** with the **touch** command.

```
touch blink.sh
```

Then, navigate to the blink.sh file through the navigator on the left side of the screen editor, open it, and fill in the following script:

```
while :
do
        /usr/bin/gpio -g toggle 6
        sleep 1
done
```

Before we proceed, let's take a closer look to the lines we just wrote. The first line **while?** created an infinite loop that runs the following two lines over and over again. The **toggle** option is an option to write the opposite condition: if the pin is Low, it will be written High, and vice versa. The **sleep 1** means to delay the program for 1 second before it continues.

➡ Can you think of a way to change the frequency of blinking by modifying one number in the above script?

After writing our script, we can press **CTRL+X** followed by confirmation by pressing the **Y** button to save it. After that, make the script executable (to give us the permission to run it with the **+x** option) with the **chmod** (change mod) command.

```
chmod +x blink.sh
```

At last, we can run the program with the bash command:h

```
bash blink.sh
```

or

```
./blink.sh
```

The LED should blink. Don't forget to stop it using **Ctrl+C**.

3.3. Additional Reference

GPIO commands

Blinking the LED tutorial

Section E

Sensors, Actuators, and Control: 1

Sensors, Actuators, and Control: 1 focuses on an introduction to how sensors take in information and translate it to data that a robot can use to perceive the world around it.

Subsection E.1
Overview

This subsection discusses the basic use of sensors, and how sensors are interpreted by computers.

UNIT E.1.1
Intro to Sensors

Explain the definition of roll, pitch, and yaw.

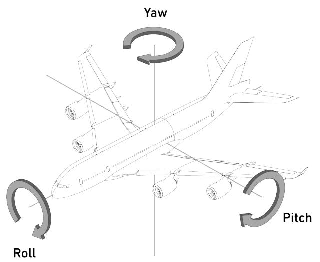


Figure 1.1. Roll, Pitch, and Yaw Diagram

Certain sensors on the drone measure Pitch, Roll, and Yaw. The IMU measures roll and pitch; The camera measures yaw.

Now, let's define some terms from the picture above.

Pitch- The rotation of the flying body around a side-to-side axis. It can be thought of as an "up and down" motion.
Roll- The motion of the flying body rocking back and forth. It can be thought of as the wings of a plane "tilting up or down".

Yaw- The rotation of the flying body along a vertical axis. It can be thought of "twisting left and right".

Reference

Answers for Class Discussion Questions

Question 1:

Answer Analog-to-Digital Converter (ADC)!

Here (Material 3.15) is what it looks like.

KNOWLEDGE AND ACTIVITY GRAPH

Voltage or Current is produced by the sensors -> amplification (convert to voltage if necessary) -> ADC

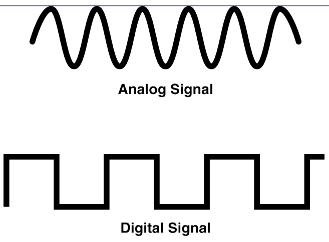


Figure 1.2. Analog and Digital Signal Diagram

Question 2:

Answer Interpolation (estimate the data points in between known data) and extrapolation (using the current trend to predict the future data)

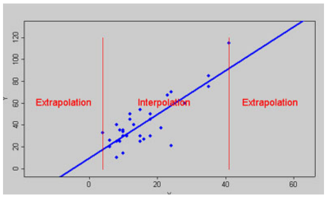


Figure 1.3. Interpolation and Exterpolation Graph

Question 3:

Answer - Filtering Frequencies: cut the frequency measurements that are unreasonably high or low, Combining data from multiple sensors, Cleverly decide which data are trustworthy

Section F
Build: 2

In this part of the build, you will be adding your first sensor to the drone – the infrared (IR) sensor. The IR sensor is used to measure distance, and it is used on the drone to measure how high it is flying. By the end of this build part, you will be able to see the output of the infrared sensor in a web browser using the drone's web interface!

Build Part 2 Instructions can be found in the Operations Manual

SECTION G
Sensors, Actuators, and Control: 2

This section demonstrates how to use the infrared sensor that you attached to your drone in Build Part 2. The first subsection demonstrates how to convert the output of the infrared sensor into a measurement of distance. The second subsection introduces the Robot Operating System, or ROS, which will allow the drone to share the distance measurement from the infrared sensor with other programs running on the Pi that make the drone fly.

SUBSECTION G.1
Sensing

Your drone needs to know how high it is above the ground if it wants to fly at a constant height. This subsection will get you a giant step closer to making this possible. The lessons in this subsection will walk through the processes of:

1. understanding the output of the infrared sensor
2. converting the output into a useful unit, such as meters which are used to measure distance
3. writing a Python program to read the sensor using the Raspberry Pi

By the end of this subsection, you will be able to measure distances with your IR sensor!

UNIT G.1.1

Calibration

1.1. Background

In the Build Part 2 Checkpoint, you read the output of your IR sensor using two methods. First, you used the multimeter to read the voltage of the IR sensor signal wire. Then, you started up the drone software and observed the IR sensor readings on the height chart. In this lesson, you will implement the steps needed to transform the voltage output in Volts to a distance in meters.

1.2. How the IR sensor works

If you take a close look at the IR sensor, you might think that it looks as if it has two eyes. One of these "eyes" is used to emit light, and the the other is used to detect light. In slightly more detail, the IR sensor works by emitting a beam of infrared light out of an infrared LED on one side of the IR sensor, and then measuring reflection using a special sensor on the other side. The sensor that measures the reflected light is called a Position Sensing Device (PSD), and it outputs a voltage that depends on where the light is hitting the sensor. If you would like a more detailed explanation, read the "How does an IR sensor work?" section of this article.

Although we know that the PSD outputs a voltage related to the position of the reflected light, we do not know what the relationship is. There are three ways that we can determine the relationship. The first is looking at the sensor *datasheet*. The second is by experimentation, and the third is by geometrical derivation.

1.3. Datasheet

A datasheet is a document that is created by the designer of some piece of technology that includes all of the information about the device. The information will vary depending on the device, but generally it will include the correct operating conditions, the limits of the device, and graphs of the devices performance.

1) Read through

Here is the datasheet for your Sharp IR sensor. Give it a brief read to become acquainted with the document, it is OK not to understand everything; some of the information is application dependent.

1. The first includes descriptions of the device, features, an some example applications.
2. The second page includes technical drawings with dimensions so that we can design our robot to accomodate the sensor.
3. The third page has information on the electrical characteristics of the sensor such as the sensor output on different surfaces.
4. The fourth page shows the timing diagram (very low-level circuit design informa-

tion) that shows how fast new sensor measurements are available.

5. The fifth page shows the relationship between the output voltage and the distance to an object. This is just what we were trying to find!

2) Looking at the graphs

Take a look at the graphs in Fig. 2. In the next section, you will be creating these graph for yourself!

Top graph The top graph shows the voltage output of the sensor at different distances. If you are given a distance, you can tell what the voltage output will be by looking at the y value of the graph. For example, if the sensor is at 60 cm, the voltage would be about 0.5 V. However, we want to know the distance given the voltage. If we were given a voltage, such as 1.0 V, there are two possible distance values on the graph: approximately 1 cm and 27 cm. If you look back at the first page of the datasheet, it says that the range of the sensor is between 10-80 cm. This means that we cannot trust the values of the sensor if it is less than 10cm away from an object. If we ignore the part of the graph before 10cm, then we know that if we are given 1.0 V from the IR sensor, then the distance must be 27cm.

Bottom graph The bottom graph show the voltage output of the sensor versus the inverse of the distance.

Q: Why did they make a graph of inverse distance?

Hint: Look at the shape of the graph

Answer: The reason this graph uses the inverse distance is because it *linearizes* the graph. In the top graph, we see that the voltage is *inversely* related to the distance. By graphing the voltage against the inverse distance, we can make the graph look like a line. The advantage of linearizing is so that we can approximate any value using the equation of a line: $y = m \cdot x + b$. The following section includes more information on linearizing and inverse relationships.

1.4. Experimental derivation

The easiest way to derive the relationship between the IR sensor voltage and distance is to experiment. For this experiment, we are going to move the IR sensor to a known distance from an object, and then we are going to record the voltage output. We will repeat these steps several times between 1cm and 100 cm. Then we will plot the data to visualize the relationship between the two quantities. We've created a spreadsheet for you to enter your data. Open this spreadsheet and click: File > make a copy.

1) Collect Data

Gather your drone build and and a meter stick. Hold the meter stick up, or lean it against a wall. Face the sensor at 0.01m (1 cm) above the ground and use the multimeter to measure the voltage, as was done in Build Part 2 Checkpoint. Record the voltage in the spreadsheet column titled “Voltage (V)”. Repeat this step for each distance measurement.

Note: For the distances less than 10cm, you will need to use the 20V setting on the multimeter when measuring. For greater distances, you can use the 2V setting.

2) Analyze the Data

Figure 1 is a scatter plot that compares the voltage output from the sensor to the distance from the ground. Take a look at one of the data points on the plot. Notice that the *x* value is the distances that you measured at, and the *y* value is the voltage output that you recorded. We can use this plot to get the distance from the voltage. However, what happens if we are given a voltage that is not one of our data points? For example, we know the voltages at 40cm and 50cm, but what if the drone is at 45cm?

3) Interpolate

Although we do not know what the voltage will be in-between our data points, we can approximate it using nearby data points. For our approximation, we can perform a *linear interpolation* between data points. Linear interpolation sounds fancy, but all it means is to play connect-the-dots with our graph; that is, we draw a straight line between each point. Figure 2 shows what the graph looks like with linear interpolation. Notice that between each point is a straight line (if the line was curved, it would not be **linear** interpolation).

Now that we know what the data is in-between points, let's try working with it again!

4) Understand the data

Now that we have interpolated between points, we can use this graph to find the distance given any voltage. Let's try it.

1. Hold your sensor above the ground at any distance between 10 and 80cm. Write down the distance somewhere.
2. Read the voltage using the multimeter
3. Look at the graph and find this voltage on the y-axis
4. Follow the graph straight down to the x-axis to find the distance that corresponds to this voltage
5. Check: was the distance that you got from the graph close to what you wrote down?

This is how the IR sensor can measure distances! But there is a bit of a problem.

Q: What happens if try to follow the same steps when the IR sensor is only 5 cm above the ground?

Answer: There are two possible distances for this voltage value! If we were only given the voltage value and asked to find the distance, we wouldn't know which one to choose!

5) Set limits

There is an easy solution to the problem we've just discovered. Let's cut out any part of the graph that has two distance values for one voltage value. It looks like if we remove all of the data before 10 cm (value may vary depending on your data), we can eliminate our problem. This is what was recommended earlier in the sensor datasheet.

Q: Do you see the disadvantage to ignoring the data below 10cm?

Answer: We cannot trust any of the sensor measurements that are less than 10cm. This is unfortunate, but since the drone will always be flying above this height, this will only be a problem during the takeoff. (The solution is to take off a little fast so that we are not flying below 10cm for long.)

The new graph of our data is shown in Figure 3. Notice that if you were to draw a horizontal line across the graph, it will never cross the voltage graph twice unlike before. This is called the “horizontal line test” and it means that all of the y-values have only one x-value. In other words, we never have the issue of choosing between two distance values. We also know that we can’t trust our measurements until the sensor is at least 10cm off of the ground.

6) Linearize the graph

With our latest graph (Figure 3), we can be given a voltage and then we can find the distance. However, it is clear that our linear interpolation is a very rough approximation of the graph. In reality, there are no straight lines between data points, the graph is curved, and if we take more data points, then we will be able to see this better. The more data points that we have, the less we have to approximate between two values. But adding more and more data points is tedious, and we don’t know how many we need to add to make the drone fly well. It would be nice if there were a better way...

And there is! The solution is to linearize the graph. Linearizing the graph means manipulating the data so that follows a line. A linear graph is easy to work with because it can be easily approximated using the equation for a line:

$$y = m \cdot x + b$$

where m is the slope (rise over run) and b is the intercept (where the graph crosses the y axis). In our case, we want:

$$\text{voltage} = m \cdot f(\text{distance}) + b$$

In the equation above, x is a function of distance:

$$x = f(\text{distance})$$

These formulas are equivalent:

$$\text{voltage} = y = m \cdot x + b = m \cdot f(\text{distance}) + b$$

Because of the shape of our data in the Figs.1 and 2, we can assume that the distance is *inversely* related to the voltage. Mathematically, an inverse relationship is defined as:

$$y \propto \frac{1}{x}$$

or

$$y = c \cdot \frac{1}{x}$$

Where the $\propto$ symbol means “proportional to”, and c is some constant.

The graph of an inverse relationship looks like:

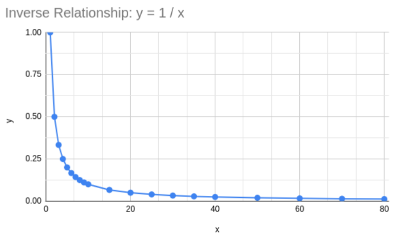


Figure 1.1. Inverse Relationship

Now that we have assumed the relationship is inverse, and have verified it by comparing the shapes of our graph and the graph of an inverse relationship, we can linearize the graph. To linearize the graph, instead of comparing voltage to distance, we can compare voltage to the inverse of the distance, or $\frac{1}{\text{distance}}$. The new graph that we get is shown in Figure 4 on the spreadsheet. Notice that our data points look like a line! We can define x as:

$$x = f(\text{distance}) = \frac{1}{\text{distance}}$$

7) Best fit line

Now that the data looks like linear, we can draw a best-fit line that is a *linear approximation* of the data. Figure 5 shows the best fit line drawn, and the legend has the equation for the line. We can find the values for m and b from the equation of the best fit line shown in Figure 5.

The one catch is that right now the line shown is

$$\text{voltage} = m \cdot \frac{1}{\text{distance}} + b$$

We need to rearrange the equation to solve for distance:

$$\text{distance} = \frac{m}{\text{voltage} - b}$$

Now we can find any distance given the voltage!

Try it out Hold your drone at a certain height between 10-80cm, read the voltage, plug it into the formula above, and see if the value you get is the correct height.

If you would like another example of linearizing graphs, watch this video tutorial

1.5. Geometrical derivation

It is possible to geometrically derive the relationship between the IR sensor output voltage and the distance to an object. Using the diagram from this article, the geometry of the sensor looks like this image:

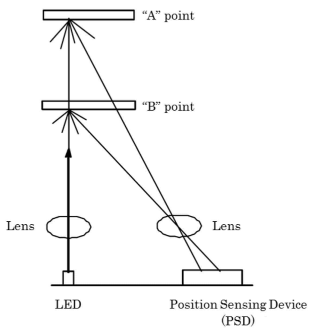


Figure 1.2. IR sensor geometry

There are a few measurements in this diagram that do not change for different distance readings. The fixed objects are the LED, the lenses, and the PSD. We can use this information to overlay triangles on top of this image for each of the measurements to simplify the geometry.

The two quantities we want to relate are the distance to the object (point A or B in the image), and the position on the PSD.

Let's create a triangle that relates the distance to an object to the position of the reflected beam. The height will be determined by the distance to an object, and the base is determined by the position of the reflected beam.

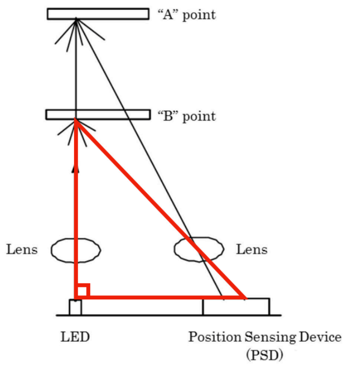


Figure 1.3. Relate Object Distance to Beam Reflection

This triangle is a *right triangle* , meaning that it contains one *right angle* (a 90 degree angle). The right angle is indicated by the box in the lower left corner. Generally, right triangles are easier to work with because one of the angles is always constrained to 90 degrees.

Let's take a look at another right triangle drawn for the object that is further away.

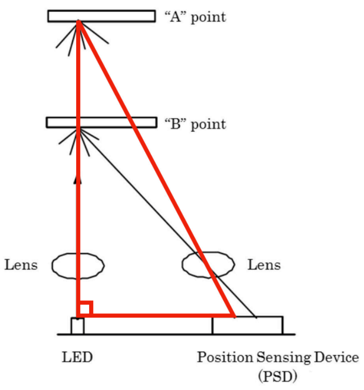


Figure 1.4. Triangle relation for further object

Q: Can you identify how the height and base changes as the object is further away?

Answer: The further away the object is, the greater the height, and the closer the reflected beam is to the left. Equivalently, as the height of the triangle increases, the length of the base decreases.

Recall that our goal is to measure the height of the triangle. We have found that there is some sort of relation between height and base; however we have no method of measuring either. What we are able to measure is the position of the reflected beam on the PSD. We also know that the distance from the PSD to the Lens in front of it is always the same. Let's draw another triangle that has a height determined by the the distance to the lens (a quantity that we can measure), and a base determined by the position of the reflected beam on the PSD (another quantity that we can measure). Let's take a look at this new triangle in addtion to the first one that we drew:

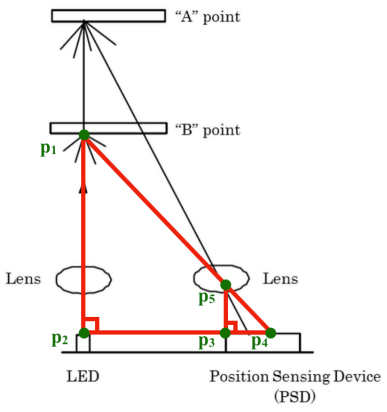


Figure 1.5. Relate Distance to Lens to Beam Reflection

This diagram has labels at each vertex. Segment p_1 to p_2 is the distance to the object. Segment p_2 to p_4 is the distance to the reflected beam from the LED. Segment p_3 to p_4 is the distance to the reflected beam from the PSD. Segment p_3 to p_5 is the distance to the lens.

Let's remove the diagram to make the geometry easier to see:

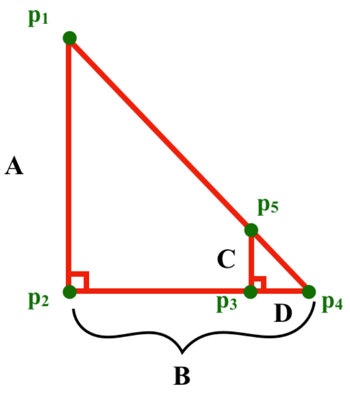


Figure 1.6. Relate Distance to Lens to Beam Reflection

In the above image, we've removed the diagram, as well as labeled the sides of the triangles:

- A (p_1 to p_2): distance to object (height of bigger triangle)
- B (p_2 to p_4): distance to reflected beam from the LED (base of bigger triangle)
- C (p_3 to p_5): distance to lens above the PSD (height of smaller triangle)
- D (p_3 to p_4): distance to reflected beam from the PSD (base of smaller triangle)

There is a special characteristic about the two triangles in the above image: they are *similar*. Similar triangles have a unique property that will help us determine the relationship between the position of the reflected beam on the PSD and the distance to the object. The property is that the ratio between corresponding sides of similar triangles are the same. This property tells us that

$$\frac{A}{B} = \frac{C}{D}$$

Now that we have one equation for our one unknown value, we can find A! Let's review what we know:

- C: fixed distance to lens
- D: Value that is given to us by the PSD
- B: fixed distance from p_2 to p_3 plus the value of D

Let's rearrange the similar triangle formula to find our unknown value:

$$A = \frac{C}{D} \cdot B$$

From this equation, we see that length of side D is inversely proportional to the length of side A. This means that the voltage reading from the PSD will be inversely related to the distance to the object. Let's take a look at the two example objects and see if this makes sense:

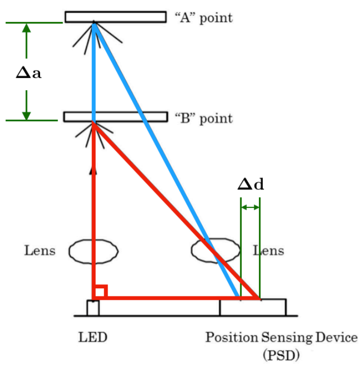


Figure 1.7. Inverse Relation

Let's say that at first our sensor was looking at "B" point in this diagram. Then, let's say that the object moved further away and is now at "A" point. We can see that the change in height, shown by Δa, of the big triangle is positive (the length increased) and relatively large; meanwhile, the change in length of the base, shown by Δd, is negative (the length decreased) and relatively small. What we've observed is that as the distance to an object increases by a lot, the distance from the PSD to the reflected beam decreases a little bit.

Based on our findings, an inverse relationship makes sense. As the object distance increases by a large amount, the reflected beam distance decreases by a small amount.

Now that we have proved geometrically, and verified analytically, that the distance to an object is inversely related to the PSD measurement, we are only a few steps away from converting the output of the IR sensor to a distance measurement in meters. Follow the experimental calibration steps to find the proportionality constant, c, of the inverse relationship.

Proof of similarity

We can prove that the two triangles are similar by proving that two angles are congruent. We know that the bottom left angles of each triangle are congruent because they are both right angles. This was easy because we drew the triangles this way. We also know that the bottom right angles of both triangles are the same because we drew the smaller triangle inside of the bigger one and both triangles share this angle. Since two corresponding angles are the same, so are the third angles. Therefore, these triangles are similar.

UNIT G.1.2
Reading Sensors

2.1. Background

The introduction to Sensors lesson explained the importance of sensors to robots, and the specific sensors used on the drone. The sensor calibration lesson demonstrated how to convert the sensor output into a useful measurement with units. Specifically, the voltage output of the IR sensor was calibrated to obtain distance measurements in meters. This calibration is important because it will allow the drone to know how high above the ground it is, which allows the user to control how high they want the drone to fly. The next step is to create a way for the drone to read these calibrated measurements so that it can use them while flying. This lesson contains the background knowledge for making a sensor and computer "talk" to each other, as well as instructions to write a python script that allows the Pi to read values from the ADC, and convert into calibrated distance measurements.

1) Analog to digital conversion

As you've learned, sensors can produce either digital or analog outputs. If the sensor produces an analog output, like the IR sensor, then it is easy for a human to read the sensor value using a multimeter; however, it is not possible for most computers, like the Pi, to read the output since they can only read digital signals. Recall that this problem was solved by introducing the analog-to-digital converter, or ADC. The new problem that arises is how to read the sensor using the Pi.

2) Communication protocols

In order for two digital devices to communicate, the device that is sending the information and the device that is receiving the information must speak the same language. For the IR sensor, the device that is sending the information is the ADC, and the device that is receiving the information is the Pi. In this case, the information is the digitized IR sensor measurement. In the world of digital devices, the "languages" are called **communication protocols**. The communication protocol used by the ADC and the Pi is called I2C (pronounced "I squared C" or "I two C"). I2C is a **serial** communinication protocol, which means that information is sent sequentially. For example, if the ADC wanted to send the binary value `1011`, it would need to first send `1`, then `1`, `0`, and finally `1`. There is lots to learn about different communication protocols, their use cases, and how they are implemented. While it can be easy to get lost in the details, it is important to keep the goal in mind: to create a way for digital devices to communicate.

3) Software libraries

Fortunately for programmers, the nitty-gritty of communication protocols are handled for us by software libraries that contain **classes** and **methods** that allow other programmers to talk to the devices. The library that will be used for the ADC is called the created by Adafruit to make it easy for programmers to use their device. Here is the github repository for this library for reference, but all you need to know is that this is what makes it easy to use the ADC.

2.2. Activity 1: Reading ADC values

Let's begin writing a Python program to read values from the ADC. We will walk through the code line-by-line, and you can either type the code, or copy and paste it.

1) Power up your Pi

Plug in the battery, connect to your drone's wifi, and to browse to its code editor: 192.168.42.1:8081.

2) Create a new directory

Create a new folder for your code in the `~/ws/src/pidrone_pkg` directory and name it `student`

3) Create a new file

Create a new file in the `~/ws/src/pidrone_pkg/student` directory and name it `ir.py`

4) Import the library

Import the Adafruit_ADS1x15 library using the Python import syntax. When you import a library, you have access to all of the functions, classes, and methods that are included.

```
# import the library
import Adafruit_ADS1x15
```

5) Create an ADS1115 instance

The Adafruit_ADS1x15 library contains a class for the ADS1115 device, which is the ADC on your drone. You will create an instance of this class using a special method called the **constructor**. Notice that for the ADS1115 class, the constructor takes in no arguments.

```
# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()
```

6) Read the ADC value

Once you've created an instance of a class (also called an **object**), you can use the methods that are defined within that class. The method that is used to read a single value from the ADC is called `read_adc`, and it takes in two arguments: the ADC gain and channel number.

Gain: If an analog signal is small, then you may want to amplify it, or multiply it by a larger number, in order to make it easier to read. The number that you multiply the signal by is called the gain. According to the ADC datasheet on Page 13, Table 3, the gain that should be used between -4.096 and +4.096 Volts is 1. Recall that the IR sensor measures between about 0 and 3 V.

Channel Number: The ADC has four input channels labeled "A0" to "A3". If you recall from Build Part 2, the yellow IR sensor wire was soldered into "A0". Therefore, the channel number you will use is 0.

Let's create **global variables** for the gain and channel number at the top of our script so that we can use these variables everytime we want to read the ADC. Global variables are values that can be referenced by name anywhere in a script. Typically, global variables are written in all capital letters.

```
# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0
```

Now that we've stored the method arguments, let's read from the ADC!

```
# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# read the adc value
adc.read_adc(CHANNEL, GAIN)
```

It doesn't do us much good just to read the value, let's store the value and print it so we can see what it is:

```
# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# read and store the adc value
value = adc.read_adc(CHANNEL, GAIN)

# print the adc value
print(value)
```

7) Run the Python Script

Now that we've written the script to read from the ADC, let's test it out!

1. Copy and paste the final code into your ir.py file.
2. Open a new terminal: Menu (top left) > Terminal > New Terminal
3. In the terminal, navigate to the student directory: `cd ~/ws/src/pidrone_pkg/student`
4. Run the script: `python ir.py`

If all went well, a number should print out in the terminal. Congrats on reading the ADC!

If you get an error, try copying and pasting the code again.

To run the script again, simply press the up arrow on your keyboard and press enter. Try running the script while holding the drones at different heights.

Q: Are there any similarities between this value and the value that you read with the multimeter?

Answer: You'll find out in activity 3!

2.3. Activity 2: Creating a Loop

It's not very fun or useful to have to re-run the `ir.py` script everytime you want to take a measurement. Since our goal is for the drone to know how high it is flying, we don't want to have to run the script each time the height of the drone changes (which is quite often!). Fortunately, there is a programming tool called a **while loop**, which lets us re-run certain lines of code while some condition holds true. Let's add a while loop to our script so that it keeps printing out the ADC output until we tell it to stop.

1) Add the while loop

The simplest condition for a while loop is the boolean, True. If we use True as the condition for the while loop, the loop will never stop, and the script will continuously read and print the ADC output.

```
# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # print the adc value
  print(value)
```

Notice that the reading and printing lines are indented. Anything that is indented after the while loop is repeated.

2) Run the Script

Follow the same steps as before to run the script now that we've updated it. When you want to stop the script, you will need to press the `control` (ctrl) key, and while holding this key down, press `c` on your keyboard.

3) Slow down!

As you'll see the values are printing out *really* fast! The while loop is running as fast as the Pi can go; unfortunately, allowing this to happen uses up the Pi's computing resources, and slows down the other processes on the Pi. How about we wait 1 second between measurements to slow down the loop; this will also make it easier for us to read the values being printed.

In order to wait 1 second in between readings, we must import the time library. Let's import this at the top of our file, next to the other import statement. Then, we can use a function in the time module to make the loop wait for one second. This function is called `sleep`, and it takes in one input, which is how many seconds you want the loop to sleep for. We'll use one second.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # print the adc value
  print(value)

  time.sleep(1)
```

4) Run the Script

Copy the changes to your `ir.py` script, and run it on your drone.

2.4. Activity 3: Navigating a Python Library

In the previous activity, you used the Adafruit_Python_ADS1x1 library to read values from the ADC. The instructions walked you through writing the code line by line. To promote self-efficacy, we will take a moment to explain how we figured out how to use this library, and how, in general, you can figure out how to use any library that you want. For example, if you later want to add a different range sensor to your drone instead of the IR sensor, you will need to find a library for the sensor and figure out how to use it.

1) Search for your hardware device

The simplest way to find a library for your device is often to read the documentation on the website you bought it from. For the ADC, this was originally purchased from Adafruit, and their webpage include instructions on how to read the sensor, and what library to use.

2) Look for examples

Once you've found a library, the next step is to look through the documentation to see if the people who wrote the library also included instructions for how to use it. These instructions are often written in a file names `README`. If there are no instructions, the next option is to look for examples, or the worst case scenario: read through all the code to figure out how it works.

A lot of learning to code involves looking at previous examples and then implementing them for yourself. Most libraries include example scripts that demonstrate the basic functionality of the library. Typically, these example scripts provide enough information to use the library and adapt it to your own needs.

For the ADC, there is no valuable README with instructions, but there are examples. If you click on this link, it will bring you to the github respository for the library. If you click on the `examples` folder, and then on the `simplest.py` file, you will see a script that demonstrates how to read each channel of the ADC, and then wait for 0.5 seconds before repeating this loop. Looks pretty similar to the script you just wrote, huh?

2.5. Activity 4: Function Composition

Now that we can continuously read the ADC output, we're one step closer to providing useful height measurements for the drone. Remember, our goal is to give the drone information about how high it is above the ground. In the previous lesson, you calibrated the output of the IR sensor to obtain a distance (in meters) from the IR sensor voltage. In this lesson, you wrote a script to read the output of the ADC. The problem that remains is how we can use your previous calibration to convert the ADC output into a height reading. The solution to this problem requires knowledge of the relationship between the ADC output and the IR sensor output, as well as a handy tool called **function composition**.

1) Relationship of ADC output to IR sensor Output

Based on how an ADC works, we know that the ADC output is **directly related** to the IR sensor output. A direct relationship means that as one value increases, so does the other. Recall that as the IR sensor moved from 10cm to 50cm from an object, the voltage decreased. Similarly, if you observe the ADC output as the IR sensor is moved from 10cm to 50cm, the value decreases. The ADC value does not have a specific unit; based on how the ADC works, the output is typically referred to as "raw ADC counts". Let's represent the relationship mathematically using the symbols a for the ADC output, c for the constant, and v for the IR sensor output:

$$v = c \times a$$

If we want to show that the voltage is a function of the raw ADC counts, we can write the equation as follows:

$$v(a) = c \times a$$

We can rearrange the equation to solve for c:

$$c = \frac{v(a)}{a}$$

2) Find the Constant

Use your multimeter and your `ir.py` script to get the value for v(a) and a. Plug these values into the above equation to find the value for c. To do this, position the IR sensor between 10-80 cm from an object, then run your ir.py script. Without moving the IR sensor, read the voltage. Divide voltage by the value printed in the ir.py script, and you've got c !

3) Function Composition

Your calibration from the previous lesson was a function of the voltage. If we can find a conversion from raw ADC counts to Volts, then we can use our our same calibration function from before!

Let's write down all of the formulas that we know:

First, your calibration from Volts to distance:

$$d(v) = m\frac{1}{v} + b$$

Second, your conversion from raw ADC counts to Volts:

$$v(a) = ca$$

Since the Pi is able to read the raw ADC counts, but our calibration is a function of Volts, we need to combine the above formulas to create a calibration function from raw ADC counts to distance. Fortunately, we can **compose** the functions to find:

$$d(v(a)) = m\frac{1}{v(a)} + b$$

By substituting the formula for $v(a)$, we are able to create a calibration function from raw ADC counts to distance!

$$d(a) = m\frac{1}{ca} + b$$

4) Dimensional Analysis

The function compositions states that the distance is a function of voltage, which is a function of raw ADC counts. Let's plug in the units to our function and use dimensional analysis to verify that the final unit is meters:

$$\text{meters} = \text{meters} \times \text{Volts}\frac{1}{\frac{\text{Volts}}{\text{raw ADC counts}} \times \text{raw ADC counts}}\text{raw ADC counts} + meters$$

After canceling the units on the top and bottom, we get meters! At this point, we have all that we need to get a distance measurement on the Pi.

2.6. Activity 5: Calibration in Code

In activity 4, we found that we can convert the raw ADC counts to distance using the function:

$$d(a) = m\frac{1}{ca} + b$$

Let's add this function to our `ir.py` script to do the conversion.

1) Define the function

To define a function in Python, you simply use the keyword, *def*, followed by the function name and parentheses that contain the function arguments. Here is the general syntax for defining a function:

```
def my_function(arg1, arg2, ..., argN):
  do something here
  do something else here
  return someValue
```

Now, define the calibration function at the bottom of your ir.py script and call it `get_distance`. It will take in one argument, `raw_ADC_counts`, and it will return the calibrated measurement

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # print the adc value
  print(value)

  time.sleep(1)
```

2) Fill in Constants

Unfortunately, this function will throw an error at this point, because neither m, c, or b are defined. Define these values before computing d by filling in the constants that you found from Activity 4 (c) and the previous lesson (m and b). If you did not do activity 4, you can use `0.00012438` for c. The variables m, c, and b are called *local variables* because they are only defined within the *get_distance* function.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0  # use your own value for m
  b = 0  # use your own value for b
  c = 0  # use your own value for c
  d = m * 1/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # print the adc value
  print(value)

  time.sleep(1)
```

3) Call the Function

Great job creating the function, now it's time to use it! Call the function in your while loop and print out the raw ADC value, as well as the calibrated measurement.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0  # use your own value for m
  b = 0  # use your own value for b
  c = 0  # use your own value for c
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  time.sleep(1)
```

4) Run the script

Copy the new code to your `ir.py` script, and run it as you did before. Use a meter stick to check if the height measurements are accurate.

Congrats on getting calibrated measurements on your drone!

SUBSECTION G.2
Middleware: ROS

This subsection introduces ROS as a tool for sharing information that is collected in one program with other programs that need the same information. For example, the Python program that you wrote in the previous lesson allows you to get distance measurements from the IR sensor. It would be useful if this information could be shared with other programs such as the controller, which tells the motors how fast to spin based on how high you want the drone to fly. Later sections will discuss the use of sensor information; this subsection describes how to share that info.

Unit G.2.1
Intro to ROS

1.1. Motivation

When writing software for a robot, it is easier to break down the software into smaller, more manageable chunks than to write one big program that does everything. While we could have a really long Python program that interfaced with each sensor, used the sensor data to estimate the state of the drone such as it's height an how fast it's moving, and then use this information to change the speeds of the motors, this has several disadvantages. First, if there is a bug in the code (a mistake in the software), it would be very hard to find in a large file where all of the code is together. Second, if someone wanted to improve just one part of the code, such as a better estimate of the IR sensor, they would need to read through a bunch of code that isn't relevant to make their change; additionally, it would be hard to undo those changes if they didn't work well. Instead, software engineers prefer to write *modular* code, where each program only performs one small job that contributes to a larger job. For example, one program can handle talking to the IR sensor, another program can interface with the camera, another can combine the data, and another can use the combined data to control the motors. Using this software design, it is easy to design and test smaller pieces of code before they go into the bigger picture. If the IR program isn't working right, it won't stop the camera from working, and it will be easy to find the problem.

When we break down the software into smaller programs, we then face the problem of making the programs talk to each other. The data collected in one program might be needed by another program. Let's consider an example of this situation on the drone.

1) Example

In the last lesson, you wrote a Python program that could read the ADC, and convert the value to a distance measurement. These steps were essential to obtain the height of the drone in meters. The problem we face now is sharing that value with other programs running on the drone. When the drone is flying, we want the *controller* to know how high the drone is, so that it can speed up or slow down the motors accordingly. We will go into details on controllers in a later section. For now, we will set up a simpler problem to solve that will demonstrate the process of using a sensor to control an actuator.

Goal: The goal is to adjust the brightness of the LED on your drone based on the distance measured by the IR sensor. If the distance is small, the LED should be bright. As the distance measurement increases, the LED should dim. This will simulate speeding up and slowing down the motors based on how high the drone is. The information in this lesson will provide the background needed to accomplish this goal in the following two lessons.

1.2. Software Architecture on the DuckieDrone

Take a look at this diagram that illustrates the software that runs on the drone. Refer back to this diagram as you read through the following points.

1) Hardware

Each rectangle with sharp corners in the diagram represents a piece of physical hardware. The IR sensor and the camera are two sensors on the drone.

2) Software

Each rectangle with rounded corners represents an individual Python program. Notice that there are specific programs that read the data from each sensor. Inside of that program, the data is read, converted into a useful measurement, and then shared with other programs. If student A's IR sensor code worked better than student B's, then student A could share just their IR code with student B, and student B would be able to use just this code with the rest of the code being their own. This is one advantage of modular software design: small parts of code can be interchanged easily.

3) Middleware

In order to share data between programs, such as the IR sensor reading, we use *middleware*. Middleware runs "between" the hardware and software on the drone. In the diagram, the middleware is represented by the arrows that connect the software.

ROS

The middleware used on the drone is called the Robot Operating System, or ROS (pronounced like "moss" but with an "r"). This is industry standard software and is used in many profession robotics companies. You can watch a few cool examples of robots that use ROS here.

ROS on the drone

For the purposes of the drone, ROS is a communication tool that allows one Python program to talk to another. For example, ROS will allow the `ir.py` program to share the distance measurement from the IR sensor with another program that can adjust how fast the motors are spinning to hold the drone at a specific height.

1.3. ROS jargon

The communication system in ROS comes with its own vocabulary. The following terms are essential for understanding ROS.

1) Node

A ROS node is simply a program that contains publishers or subscribers. On the drone, a node is just one of the Python programs that is used to make the drone fly.

For more info: ROS wiki link

2) Publishers

A publisher is a ROS entity that sends messages to a topic. In python, a publisher is just an object with a "publish()" method that sends the message.

For more info: ROS wiki link

3) Messages

A message is a structured format for data. Messages standardize how the publisher is supposed to format the data, so that the receiver knows how to read it. For example, there is a Boolean message type where the data can only be `True` or `False`. There is also a String message type, where the data can be any text such as `this is my message!`. We can also make custom message types that combine two or more existing message types. For example, we could create a message type that contains two Booleans and one String.

```
bool
bool
string
```

We would want to label the Booleans and the String so that the sender and receiver would know which data was which. If the first Boolean determined whether I wanted ice cream, and the second Boolean determined whether I wanted to pay for it, I would not want those mixed up! Here is what a custom message type could look like:

```
bool      do_i_want_icecream
bool      do_i_want_to_pay_for_it
string    flavor
```

The first column shows the type of the data, also known as the **fieldtype**. The second column shows the label for the data, also known as the **fieldname**. Together, the fieldtype and fieldname make one **field** in a ROS message.

The way that we would use this message in Python is to create a blank message, and then to edit the fields. For example:

```
my_icecream_order = OrderMessage()  # create a blank message

OrderMessage.do_i_want_icecream = True
OrderMessage.do_i_want_to_pay_for_it = False
OrderMessage.flavor = "Strawberry"
```

Now I can publish my_icecream_order and my ice cream robot will go and buy me strawberry ice cream!

For more info: ROS wiki link

4) Topic

A ROS topic is a named bus that messages get published to and subscribed to. In Python, a ROS topic is just a String that identifies where a publisher should send messages, and where a subscriber should look to receive those messages. Only one message type can be sent along a topic, and this is set when the topic is created by a publisher.

For example, we might want to create a topic called `order` that has the message type `OrderMessage`. We can create a publisher that publishes our ice cream orders to the `order` topic.

For more info: ROS wiki link

5) Subscriber

A ROS subscriber "subscribes" to a topic and listens for messages. When the subscriber is created, it is told what topic to listen to. As soon as a subscriber receives a message, it passes along the message to a **callback** function.

For more info: ROS wiki link

6) Callback

A callback is a function that takes as input a ROS message, and then performs some action based on that message. In some cases, the action might just be storing the message data. In other cases, the action might involve the physical world. In the ice cream example, the callback function would hopefully make our ice cream robot go and buy us ice cream. In the case of the drone, the flight controller node is subscribed to roll, pitch, yaw, and throttle commands on the `/pidrone/fly_commands` topic. When new commands are published to this topic, the callback in the flight controller subscriber sends these commands to change the speeds of the drone motors.

7) Master

In order for ROS nodes to communicate with messages over topics, each node must register with the ROS Master. The ROS master facilitates communication between nodes by keeping track of all of the nodes, topics, publishers, and subscribers. This information is used to help nodes find each other so that they can communicate.

For more info: ROS wiki link

8) Summary

ROS allows a *node* to *publish* a *message* to a *topic* and then another node can *subscribe* to the topic to receive messages. Once a message is received, the subscriber passes the message to a *callback* function that does something with the data. This communication is facilitated by a *ROS Master*, which allows nodes to find each other to communicate.

1.4. ROS commands

There are a number of useful ROS commands that you can use in the terminal.

For details on all of the ROS commands: ROS wiki

1) roscd

`roscd` is the ROS version of `cd`; it lets you navigate directly to ROS packages. For example, if you are in the home directory on the drone, instead of typing: `cd ~/ws/src/pidrone_pkg`, you can just type `roscd pidrone_pkg`.

2) roscore

`roscore` is the command used to start a ROS master. No arguments are needed to run this command

3) rostopic list

`rostopic list` will list all of the ROS topics that are currently being published and subscribed to

4) rostopic info

`rostopic info (topic_name)` will list the publishers, subscribers, and message type of whatever topic you replace `(topic_name)` with

5) rosbag

rosbag allows you to record all of the messages sent to topics. you can specify just one topic to record using `rosbag record (topic_name)`, or you can record all of the topics using `rosbag record -a`. Recording data is useful for testing and debugging. You can also use `rosbag play (file.bag)` to play back the recorded data.

For more info: ROS wiki

1.5. Environment Variables

Environment variables modify key attributes of a program. For our applications in ROS, the most important environment variable is `ROS_MASTER_URI`. To check the value of this environment variable, type `` echo $ROS_MASTER_URI` and press enter in the terminal, and the value will be printed out. the `$ `` symbol is used to signal environment variables. To change an environment variable, you can either type `export (VARIABLE)=(value)` with the correct variable and values into the terminal. These changes will only be temporary and you'll need to export the values every time. Alternatively, you can create a shell script to export the values for you, as we have done in the pidrone_pkg. Take a look at the `setup_for_managed_mode.sh` and `setup_for_master_mode.sh` scripts.

1) ROS MASTER URI

The `ROS_MASTER_URI` environment variable sets the IP address and port of the ROS master. All of the ROS nodes will look to this address to register themselves. If there are two computers both connected on the same internet network and both running ROS, one computer can start up a ROS Master using `roscore`, and the other computer can set it's `ROS_MASTER_URI` to be the same as the first computer's. Then, the ROS nodes on the two separate computers can talk to each other! This is a very useful feature because it allows the computation load to be distributed across devices. For example, some of the programs that are run on the drone can instead be run offboard on another computer running ROS. This is necessary for some of the features on the drone. However, nearly all of the DuckieDrone's programs can be run onboard.

1.6. Catkin workspace

There is one more essential ROS element that is worth mentioning if you are interested in creating your own workspace. In programming, a workspace is used to keep track of all of the programming files, dependencies, and executable code. For ROS, the workspace setup is called catkin. The main thing to note is that ROS packages (a set of programs that run together), go in the `src` directory. This is why the path to the `pidrone_pkg` is `~/ws/src/pidrone_pkg`. For instructions on creating your own workspace, refer to the ROS wiki.

UNIT G.2.2
Creating a ROS Publisher

2.1. Flashback to the sensor reading code

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0  # use your own value for m
  b = 0  # use your own value for b
  c = 0  # use your own value for c
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  time.sleep(1)
```

2.2. Creating a ROS publisher

We are going to build upon the above code to write a ROS publisher that receives the distance from sensor and publishes it.

1) Import the ROS python library and the needed message types.

We are going to use the ROS python library to write the publisher, and, as introduced in the lesson, the floating point message type as well.

- ☀ Add the line `import rospy` to the top of the program.
- ☀ Add the line `from std_msgs.msg import Float32` to the top of the program.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# import the ROS python library
import rospy

# import the Float32 message type
from std_msgs.msg import Float32

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0
  c = 0
  b = 0
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  time.sleep(1)
```

2) Create a publisher

We follow the syntax to create a ROS publisher.

☀ Type `distance_publisher = rospy.Publisher('distance', Float32, queue_size=1)` right before the global variables at the top. The first argument is the topic name, the second is the message type, the third serves to limit the number of queued messages in case the subscriber is not receiving fast enough. For the purpose of this assignment, we can just set it to 1.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# import the ROS python library
import rospy

# import the Float32 message type
from std_msgs.msg import Float32

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# The ROS publisher that publishes the distance
distance_publisher = rospy.Publisher('distance', Float32, queue_size=1)

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0
  c = 0
  b = 0
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  time.sleep(1)
```

3) Create an infrared sensor node

It is important to tell rospy the name of your ROS node, otherwise it would not be able to communicate with the ROS Master.

☀ Type `rospy.init_node('infrared_node')` in the next line.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# import the ROS python library
import rospy

# import the Float32 message type
from std_msgs.msg import Float32

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# The ROS publisher that publishes the distance
distance_publisher = rospy.Publisher('distance', Float32, queue_size=1)

# Initiate the IR sensor node
rospy.init_node('infrared_node')

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0
  c = 0
  b = 0
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  time.sleep(1)
```

4) Publish the message in the previously created while loop

☀ Type `distance_publisher.publish(Float32(distance))` right after the line `print(distance)` in the while loop.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# import the ROS python library
import rospy

# import the Float32 message type
from std_msgs.msg import Float32

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# The ROS publisher that publishes the distance
distance_publisher = rospy.Publisher('distance', Float32, queue_size=1)

# Initiate the IR sensor node
rospy.init_node('infrared_node')

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0
  c = 0
  b = 0
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while True:

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  # publish the calibrated value
  distance_publisher.publish(Float32(distance))

  time.sleep(1)
```

5) Change the structure of the while loop to a ROS manner. (This step is just a change of syntax)

- ☀ Change the top of the loop from `while True:` to `while not rospy.is_shutdown():`

- ☀ Change the bottom of the loop from `time.sleep(1)` to `rospy.sleep(1)`.

```
# import the time library
import time

# import the library
import Adafruit_ADS1x15

# import the ROS python library
import rospy

# import the Float32 message type
from std_msgs.msg import Float32

# create an instance of the ADS1115 class
adc = Adafruit_ADS1x15.ADS1115()

# The ROS publisher that publishes the distance
distance_publisher = rospy.Publisher('distance', Float32, queue_size=1)

# Initiate the IR sensor node
rospy.init_node('infrared_node')

# global variables
GAIN = 1
CHANNEL = 0

# define the calibration function
def get_distance(raw_ADC_counts):
  m = 0
  c = 0
  b = 0
  d = m * 1.0/(c * raw_ADC_counts) + b
  return d

# loop to continuously read and print the ADC output
while not rospy.is_shutdown():

  # read and store the adc value
  value = adc.read_adc(CHANNEL, GAIN)

  # call the calibration function
  distance = get_distance(value)

  # print the calibrated value
  print(distance)

  # publish the calibrated value
  distance_publisher.publish(Float32(distance))

  rospy.sleep(1)
```

Unit G.2.3
Subscriber, PWM and Open Loop

In this section we will introduce you to the concepts of PWM, Pulse Width Modulation, building an open loop controller, and writing a ROS subscriber. We will use these techniques to control the brightness of your LED based on the ROS Distance Publisher you just wrote.

3.1. PWM

We begin with a discussion of Pulse Width Modulation. PWM, is the technique used to control the brightness of an LED, the speed of your drones DC motors, or any type of control where you need to produce an analog type output with digital means. The Raspberry pi GPIO output pins produce a square wave output, the pins gives us either 3.3V (when turned HIGH) or 0V (when turned LOW). If we wanted to dim an LED with an analog output we could just send half as much voltage, but this option does not exist with the digital outputs on the Rasberry Pi GPIO. Instead what we must do is turn the signal on and off very quickly. If adjust the speed at which the ON and OFF signals are sent then the brightness of the led will be changed. It would be prudent to discuss the terms associated with PWM.

1) TON (On Time): Is the time the signal is high.

2) TOFF (Off Time): Is the time the signal is low.

3) Period: Is the sum total of on time and off time.

4) Duty Cycle: It is the percentage of time when the signal was high in the time of the period.

5) Frequency: Is the number of cycles per second (measured in Hz)

Let’s consider running the LED at a 50% duty cycle at 1Hz. The LED would stay on for half a second and would be off for half a second. This would appear as a blink to the human eye. But if we increase this frequency to 100Hz (100 cycles per second) at a 50% duty cycle, the blinking becomes imperceptible to the human eye. Instead the LED appears to be glowing at half brightness.

3.2. Open Loop Controllers

An Open-loop control system, also called a non-feedback system, is a continuous control system in which the output has no influence of the input signal. Put into other words, in an open-loop control system the output is not measured and is not provided as “feedback” for the input. Instead an open loop system uses a single input command or set point.

A disadvantage of this is the open-loop system has no knowledge of the output condition thus it can't correct any errors if the preset value drifts, even if this results in large deviations from the preset value. Because of this lack of measurement of output another disadvantage of open-loop systems is that they are poorly equipped to handle disturbances or changes in external conditions that may affect the output.

Another type of controller is the closed loop controller which we will discuss in the PID control lesson.

A good example of an open-loop system is a timed clothes dryer. The timed clothes dryer will continue to apply heat to wet clothing for the duration of its input even if those clothes are already dry.

In this lesson we will create an open loop controller based on the distance topic you wrote in the previous lesson to control the brightness of your LED.

3.3. Writing a ROS Subscriber:

Section H
Build: 3

In this phase of the build, you'll be adding the essential elements of every drone– the motors, ESCs, and the flight controller. By the end of this build part, you will have a configured flight controller, and will spin the motors (without the props on).

Build Part 3 Instructions can be found in the Operations Manual

SECTION I
Sensors, Actuators, and Control: 3

This section describes how the actuators on your robot (the motors) work, as well as the sensor that appears on all drones, and many other robots: the inertial measurement unit, or IMU.

SUBSECTION I.1
Motors

This subsection covers the functionality of brushless motors, and describes how they can be controlled to make your drone fly.

Unit I.1.1

Intro to the Motors

Subsection I.2
IMU

This subsection explains the two sensors that make up the IMU, as well as how these sensors are used to estimate the roll, pitch, and yaw of the drone.

Unit I.2.1
Intro to the IMU

1.1. Applications

IMUs have many applications. Even cars use IMUs to measure the acceleration to determine when it is time to deploy the airbags. Some computer hard drives use IMUs to detect if the hard drive is falling, and turn off to protect the data before the crash occurs. The IMU that is used on your drone originated in a gaming console used to detect hand motions– can you guess which one? The answer is the Nintendo Wii remote, and the IMU is the sensor that allows you to swing a tennis racket, or get a strike in virtual bowling.

SECTION J
Build: 4

In this section of the build, you will attach the camera, and finalize the drone assembly. By the end of this build part, you will be ready to fly!

Build Part 4 Instructions can be found in the Operations Manual

Section K
Closed Loop Control

This section introduces feedback control systems, and describes the most commonly used feedback controller: the proportional integral derivative, or PID, controller. Although the mathematics behind this controller involve calculus, the high level understanding of the controller is actually quite intuitive. By the end of this section, you will program your own PID controller!

Subsection K.1
PID Controller

Module Overview:

A PID (proportional, integral, derivative) controller is a control algorithm extensively used in industrial control systems to generate a control signal based on error. The error is calculated by the difference between a desired setpoint value, and a measured process variable. The goal of the controller is to minimize this error by applying a correction to the system through adjustment of a control variable.

Unit K.1.1
Intro to PID

1) Open Vs Closed Loop Systems

☀ Open loop system, also known as a non-feedback system, is a continuous system where output does not affect the control action of the input (Electronics Tutorials). Ex: Toaster

☀ A closed loop system, also known as a feedback system, is a system where the control action is based on the output (Electronics Tutorials). Ex: Body thermoregulation

2) PID Terms

Process Variable, represented by $y(t)$: The parameter of the system that is being monitored and controlled.

Setpoint, represented by $r(t)$: The desired value of the process variable.

Control Variable, represented by $u(t)$: The output of the controller that serves as input to the system in order to minimize error between the setpoint and the process variable.

Steady-State Value: The final value of the process variable as time goes to infinity.

Steady-State Error: The difference between the setpoint and the steady-state value.

Rise Time: The time required for the process variable to rise from 10% to 90% of the steady-state value.

Settling Time: The time required for the process variable to settle within a certain percentage of the steady-state value.

Overshoot: The amount the process variable exceeds the setpoint, and it is expressed as a percentage.

3) Example relating to biological system:

These terms are applicable to a biological system that we rely in. Thermoregulation relies on a negative feedback system.

The process variable: your body temperature.

Setpoint: Regular body temperature is 98.6 degrees F (or 37 degrees C).

Control Variable:

If you are cold, and you have a lower body temperature than usual, your body will generate more heat to restore back to regular body temperature.

If you are warm, and you have a higher body temperature than usual, your body will reduce heat in your body to restore back to regular body temperature.

4) The General Algorithm

The error of the system $e(t)$, is calculated as the difference between setpoint and proces variable.

$$e(t) = r(t) - y(t)$$

The controller aims to minimize the rise time and settling time of the system, while eliminating steady-state error and maximizing stability (no unbounded oscillations in the process variable). It does so by changing the control variable $u(t)$ based on the three control terms.

UNIT K.1.2

The Three PID Terms

Here is a helpful intro video explaining PID controllers

1) The Three PID Term Definitions

The Proportional Term

The proportional term produces an output that is proportional to the calculated error:

$$P = K_p e(t)$$

During programming, you would represent this as:

$$P = K_p e(t_k)$$

The magnitude of the proportional response is dependent upon K_p which is the proportional gain constant. A higher proportional gain constant indicates a greater change in the controller's output in response to the system's error.

The propellers will spin faster the farther away the drone is.

The Derivative Term

The derivative term is determined by the rate of change of the system's error over time multiplied by the derivative gain constant K_d.

In terms of calculus, it would be represented like this:

$$D = K_d \frac{de(t)}{dt}$$

In terms of programming and without the use of calculus, it can be represented by:

$$D = K_d \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$

The derivative term will do something like: propellors will spin slower the faster the drone is moving. Or a similar example is when drag is pulling harder, the faster the drone is moving.

The Integral Term

The integral term accounts for the accumulated error of the system over time. The output produced is comprised of the sum of the instantaneous error over time (or simply the instant rate of change in the error) multiplied by the integral gain constant K_i.

In terms of calculus, the equation would look like this:

$$I = K_i \int_0^t e(\tau) d\tau$$

With no calculus, the equation would look like this:

$$I = K_i \sum_{i=0}^{k} e(t_i)\Delta t$$

The integral term will do something like: propellors will spin faster the longer the drone is away from the desired set point.

2) The Overall Control Function

The overal control function consists as the sum of proportional, integral, and derivative terms.

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int_0^t e(\tau)d\tau$$

The figure below summarizes the inclusion of a PID controller within a basic control loop.

Figure 2.1. PID Controller Block Diagram

3) Tuning:

Tuning a PID controller is done by setting the conotrl parameters K_p, K_i, K_d to values that fit to be able to get an ideal control response. The three control terms may be correlated and so changing one parameter may impact the influence of another. The general effects of each term are therefore useful as reference, but the actual effects will vary depending on the specific control system.

4) Altitude PID in Simulation

In this part of the project, you will be implementing a PID controller for a simulated drone that can only move in one dimension, the vertical dimension. You can control the speed the motors spin on the drone, which sets the thrust being generated by the propellers. In this system, the process variable is the drone's altitude, the setpoint is the desired altitude, and the error is the distance in meters between the setpoint and the drone's altitude. The output of the control function is a PWM (pulse-width modulation) value between 1100 and 1900, which is sent to the flight controller to set the drone's throttle.

To run the simulation, you need to use the vnc server. You can find the installation link here.

Run `sudo vncserver`.

With bash, navigate to the file named *drone_simulator*, which is located within the scripts folder of pidrone_pkg-master folder.

You should implement the discretized version of the PID control function in *student_pid_class.py*:

$$u(t) = K_p e(t_k) + K_i \sum_{i=0}^{k} e(t_i)\Delta t + K_d \frac{e(t_k) - e(t_{k-1})}{\Delta t} + K$$

$$K_p, K_i, K_d, K = Constants\ and\ Offset\ Term$$

$$e(t_k) = Error\ at\ Time\ t_k$$

$$\Delta t = Time\ Elapsed\ from\ Previous\ Iteration$$

Notice that there is an extra offset term K added to the control function. This is the base PWM value/throttle command before the three control terms are applied to correct the error in the system.

To tune your PID, set the parameters (K_p, K_i, K_d, K) in *z_pid.yaml.*

To test your PID, run `python sim.py` on your base station or a department computer but not on your drone, since it requires a graphical user interface to visualize the output. The PID class in *student_pid_class.py* will automatically be used to control the simulated drone. The *up* and *down* arrow keys will change the setpoint, and *r* resets the simulation.

You will need *numpy*, *matplotlib*, and *yaml* to run the simulation. To install these dependencies, run `pip install numpy matplotlib pyyaml`.

Effects of K_p:

Increasing K_p will proportionally increase the control output. This causes the system to react more quickly (thereby decreasing the rise time and the settling time by a small amount). Even so, setting the proportional gain too high could cause massive overshoot, which in turn could destabilize the system. Increasing K_p also has the effect of decreasing the steady-state error. However, as the value of the process variable approaches the setpoint and the error decreases, the proportional term will also decrease. As a result, with a P-controller (a controller with only the proportional term), the process variable will asymptotically approach the setpoint, but will never quite reach it. Thus, a P-controller cannot be used to completely eliminate steady-state error.

Effects of K_i:

The integral term takes into account past error, as well as the duration of the error. If error persists for a long time, the integral term will continue to accumulate and will eventually drive the error down. This has the effect of reducing and eliminating steady-state error. However, the build-up of error can cause the value of the process variable to overshoot, which can increase the settling time of the system, though it decreases the rise time.

Effects of K_d:

By calculating the instantaneous rate of change of the system's error and using this slope for linear extrapolation, the derivative term anticipates future error. While the proportional and integral terms both act to move the process variable to the setpoint, the derivative term seeks to dampen their efforts and decrease the amount the system overshoots in response to a large change in error (which would greatly affect the magnitude of the proportional and integral contributions to the overall control output). If set at an appropriate level, the derivative term reduces oscillations, which decreases the settling time and improves the stability of the system. The derivative term has negligible effects on steady-state error and only decreases the rise time by a minor amount.

Figure 2.2. General Effects of Control Terms

UNIT K.1.3
Implementation of PID

1) Altitude PID in Simulation

In this part of the project, you will be implementing a PID controller for a simulated drone that can only move in one dimension, the vertical dimension. You can control the speed the motors spin on the drone, which sets the thrust being generated by the propellers. In this system, the process variable is the drone's altitude, the setpoint is the desired altitude, and the error is the distance in meters between the setpoint and the drone's altitude. The output of the control function is a PWM (pulse-width modulation) value between 1100 and 1900, which is sent to the flight controller to set the drone's throttle.

To run the simulation, you need to use the vnc server. You can find the installation link here.

Run `sudo vncserver`.

With bash, navigate to the file named *drone_simulator*, which is located within the scripts folder of pidrone_pkg-master folder.

You should implement the discretized version of the PID control function in *student_pid_class.py*:

$$u(t) = K_p e(t_k) + K_i \sum_{i=0}^{k} e(t_i)\Delta t + K_d \frac{e(t_k) - e(t_{k-1})}{\Delta t} + K$$

$$K_p, K_i, K_d, K = Constants\ and\ Offset\ Term$$

$$e(t_k) = Error\ at\ Time\ t_k$$

$$\Delta t = Time\ Elapsed\ from\ Previous\ Iteration$$

Notice that there is an extra offset term K added to the control function. This is the base PWM value/throttle command before the three control terms are applied to correct the error in the system.

To tune your PID, set the parameters (K_p, K_i, K_d, K) in *z_pid.yaml*.

To test your PID, run `python sim.py` on your base station or a department computer but not on your drone, since it requires a graphical user interface to visualize the output. The PID class in *student_pid_class.py* will automatically be used to control the simulated drone. The *up* and *down* arrow keys will change the setpoint, and *r* resets the simulation.

You will need *numpy*, *matplotlib*, and *yaml* to run the simulation. To install these dependencies, run `pip install numpy matplotlib pyyaml`.

3.1. Problem 1: Implement an Idealized PID

Exercises

1. Implement the step method to return the constant K. At what value of K does the drone takeoff? What could happen if K were set too high on a real drone? Set K to 1300 for the remainder of the questions.
2. Implement the P term. What happens when the absolute value of K_p is very large? What happens when its absolute value is very small? Can you tune the P term to stop oscillations? Why or why not?
3. Implement the D term. Set K_p to zero. What happens when K_d is 50? 500? 5000?
4. Now tune K_p and K_d so that the drone comes to a steady hover. Describe the trade-off as you change the ratio of K_p to K_d. Can the drone stabilize at its target (zero steady-state error)? Why or why not?
5. Implement the I term and observe the difference between PD and PID control. What role does the I term play in this system? What happens when K_p and K_d are set to zero?
6. Implement the reset method and test its behavior. If implemented incorrectly, what problems can you anticipate reset causing?
7. Finally, tune the constants in your PID controller to the best of your abilities. When the setpoint is moving, the drone should chase the setpoint very closely. When the setpoint is still, the drone should converge exactly at the setpoint and not oscillate. Report your tuning values.

3.2. Problem 2: Tuning a PID with Latency

Now, we introduce latency! Run the simulation as `python sim.py -l 6` to introduce 24 milliseconds of latency (six steps of latency running at 25 hz).

Exercises

1. Tune the constants in your PID controller to the best of your abilities. The drone should chase the setpoint very closely, but will converge more slowly when the setpoint is still. Report your tuning values.
2. Compare your tuning values to the values you obtained in problem 1.
3. Explain the effect of latency on each control term.

3.3. Problem 3: Tuning a PID with Latency, Noise, and Drag

In the most realistic mode, you will tune a controller with latency, noise, and a drag coefficient. You can do this with the command line arguments `python sim.py -l 3 -n 0.5 -d 0.02` to be most realistic to real-world flight.

Exercises

1. Tune with these arguments to be as good as possible. Report your tuning values.
2. Compare your tuning values to the values from problems 1 and 2.

Run `python sim.py -h` to see the other simulator parameters. We encourage you to experiment with those and observe their effects on your controller.

1) Tuning 1D Controls: Part 1: Planar Tuning

In this portion of the project, you will be tuning the low rate integral terms of the PID controllers that we've provided.

3.4. Trimming your Drone

Due to differences in the weight distribution and other factors that cause asymmetries, the drone will tend to initially drift in a particular direction. In order to tune your altitude PID, the planar motion of the drone needs to be controlled. This is important so that the drone does not fly uncontrollably across the room while you're trying to tune its altitude controller. To control the drone's planar motion while you're tuning the altitude, we've created and tuned PIDs to do this for you, but you will need to tune the initial low-rate integral terms to account for the uneven weight distribution specific to your drone. You will first use the provided altitude PID to tune the planar controllers, and then you will tune your altitude PID with the tuned planar controllers.

Write brief answers to all exercises in *answers_pid.md*.

3.5. Problem 1: Understanding the Controller

Our controller is a dual I-term (integral term) PID controller. The high-rate I-term changes quickly, allowing fast response to changes. The low-rate I-term changes slowly, allowing the drone to adjust to systemic sources of error. The provided PID gains have been pretuned to this drone hardware, and should not need significant modification for your specific drone. But, the initial low I-terms do need to be adjusted based on the static error of your specific drone.

Exercises
1. Name a source of static error that the low-rate I term can correct for.
2. Name two sources of dynamic error that the high-rate I term can correct for.

3.6. Problem 2: Tune the Throttle

The first step in the tuning process is finding an initial throttle value that allows your drone to have a smooth and controlled takeoff. To do this, you'll be adjusting the value of `throttle_low.init_i` in *pid_class.py*. This is the initial value of the low-rate (slow changing) integral term for the throttle, which controls altitude. The default value is 100. you will tune this value by having the drone take off, observing its behavior, and modifying the value accordingly. Each time you wish to change the value, you will need to restart *pid_controller.py* to use the new value.

Setup
1. Prepare your drone to fly over a highly textured planar surface1.
2. Navigate to `4 of the screen.
3. Quit the program by pressing ctrl-c.

Exercises
1. In this screen (`4), use a text editor (such as vim or nano) to modify `throttle_low.init_i` in *pid_class.py* to test out different values for `throttle_low.init_i`.

Be cautious when modifying this value because the drone could take off abruptly with a value that is too high. The specific `throttle_low.init_i` value is drone specific, but typical values range between 50 and 150. Try both of these values and two more values between then. In one sentence, describe the drone's behavior as a result of changing the value up and down.
2. Now find the value for which your drone is able to have a smooth and controlled takeoff. The goal is to reduce the overshoot and undershoot for the drone to takeoff and fly stable at 0.3m. Try changing this value in increments of 10 and then 5 until you find a value that allows the drone to take off at a reasonable rate. Record this value in your answers.

3.7. Problem 3: Set the Trim

Next you will set the trim on roll and pitch. You will do this by tuning the low I-terms to adjust for the static errors that exist on your drone. The default value is 0, and positive values will move the drone to the right or forward, and negative to the left or backward, depending on the axis you're modifying. Note that you may need to repeat this process periodically, for example after a crash or the like. When performing this process, each time make sure that you:

- Place the battery in the same place each time as much as possible so the weight is distributed the same.
- Plug the flight controller while the struts are fully engaged and the drone is level, so the gyros are well calibrated.
- Always place the drone so that the camera is closer to you and the skyline is farther away.

Setup
Modify *pid_controller.py* to print out the low rate integral terms of the PIDs by finding the block of code shown below and uncommenting the following print statements

```
print 'roll_low.init_i', pid_controller.pid.roll_low.init_i
print 'pitch_low.init_i', pid_controller.pid.pitch_low.init_i
```

You will also need to set the `verbose` variable in this file to zero so that these print statements will not be overridden by the other print statements: `verbose = 0`

While flying, the low-rate I-terms will change to account for the static flight error, and when you disarm the drone, the initial low-rate I terms will be set to these changed values, thus allowing the low-rate I terms to start at this corrected value. Eventually, these values will converge, and your drone will no longer drift. Once converged, you will save the values by modifying the variables `self.roll_low.init_i` and `self.pitch_low.init_i` in `pid_class.py` to the corresponding value printed in `4 of the screen after disarming. This will store the initial low-rate I-terms between flights.

Exercises

1. Perform one flight. After the drone takes off, do not give it movement commands but allow it to drift.
2. Disarm the drone before it flies too far in any direction.

3. Write down the low-I values printed in `4 of the screen.
4. Pick up and move the drone by hand back to the center of the flying area.
5. Repeat steps 1-4 until the values that are printed out after disarming have converged (roughly when the change in magnitude is less than 1).
6. Once these values have converged, record these values in your answers.

Footnotes

1A flat posterboard scribbled or written on with marker will work.

1) Tuning 1D Controls: Part 2: Altitude Tuning

In this part, you will be transferring the altitude PID you created in part 1 onto your drone. You will then tune the PID gains on your drone as you did in the simulator.

3.8. Problem 1: Flying with Your Altitude PID!

Now that the planar PIDs are tuned, and you have found a value for `throttle_low.init_i` that allows the drone to take off at a reasonable rate, you will be using your altitude PID to control the height of the drone. To tune your altitude PID, you will first use the Ziegler-Nichols tuning method to generate an initial set of tuning parameters. You will then fine tune these parameters similar to how you tuned the drone in simulation.

To use your PID, you'll be running *student_pid_controller.py* instead of *pid_controller.py*. This will allow your PID to run alongside our planar PIDs, and on top of our throttle low-rate I-term which you found previously. Your PID will be responsible for keeping the drone flying steady vertically.

Setup

Change directories to `~/ws/src`. Run `git clone https://github.com/h2r/project-pid-yourGithubName.git`. In your repo, change "pidrone_project3_pid" to "project-pid-yourGithubName" in *package.xml* and "project(pidrone_project3_pid)" to "project(project-pid-yourGithubName)" in *CMakeLists.txt*. Also remove the msg folder, and comment out "add_message_files" in *CMakeLists.txt*. Then change directories back to `~/ws/` and run `catkin_make --pkg project-pid-yourGitHubName`.

OR

Use the `scp` command to transfer *student_pid_class.py*, *student_pid_controller.py*, and *z_pid.yaml* from the repo on your base station to the scripts folder of your drone (`~/ws/src/pidrone_pkg/scripts/`). In the instructions below, instead of using `rosrun`, you may use `python` to execute your scripts.

Change directories into `~/ws/src/pidrone_pkg` and modify *pi.screenrc* to start up with your altitude pid by changing `python pid_controller.py\n` to `rosrun project-pid-yourGitHubName student_pid_controller.py\n`. Prepare your drone to fly and then navigate to `4 of the screen. Press ctrl-c to quit student_pid_controller.

In this screen (`4), modify `~/ws/src/project-pid-yourGitHubName/z_pid.yaml` by setting $K$ to 1250 and the rest of the gain constants to 0. Now run `rosrun project-pid-yourGitHubName student_pid_controller.py` to fly with your altitude PID.

Exercises

Fly your drone and observe its flight. Tune K_p by slowly increasing its value between flights until you can see the drone moving up and down with uniform oscillations. Each time you will need to quit the controller, edit `~/ws/src/project-pid-your-GitHubName/z_pid.yaml`, and then run `rosrun project-pid-yourGitHubName student_pid_controller.py` again to use the new PID gains.

1. Record your final K_p value that causes uniform oscillations as K_u, the ultimate gain.

2. Fly your drone and pause the altitude graph on the web interface when you see two peaks. Find the time difference between these two peaks and record this value as T_u, the ultimate period.

3. Use your K_u and T_u values to compute K_p, K_i, and K_d. Refer to the equations in the Ziegler-Nichols section in the introduction to this project. Record these values and change *z_pid.yaml* accordingly.

4. Fly your drone with the set of tuning values generated by the Ziegler-Nichols method. Note that the Ziegler-Nichols method should enable safe flight, but will probably not control your drone's altitude very well! Empirically tune the gain constants in *z_pid.yaml* on your drone as you did in the simulator portion of this project. 2 Record your final tuning values.

Take a video of your drone flying first using our altitude pid by running *pid_controller.py* in `4, then take a video of your tuned pid by running *student_pid_controller.py* in `4. See if you can get yours to track the altitude setpoint better than ours! The drone should get to the setpoint quickly and stay there without bouncing up and down.

Footnotes

2 Use the graph on the web interface to observe the drone's behavior as it oscillates around the 0.3m setpoint the drone's ability to hover at the setpoint. When observing the drone itself, try to get eye-level with the drone to just focus on the the altitude and ignore the planar motion; it is easier to focus on one axis at a time when tuning the PIDs. The planar axes can be re-tuned after you tune your altitude pid if need be.

1) Tuning 1D Controls: Part 3: Position Control

Thus far, the planar PIDs have been used to control the velocity of the drone; now, you will use cascaded PIDs to control the position of the drone. The cascaded PIDs are set up so that the position controller forms the outer loop which uses the position error to provide setpoint velocities for the inner loop velocity controller.

2) How to Engage Position Control

Engaging position control involves two steps. First you have to tell the drone to "remember" a frame. You can do this using the *r* key. This will save the frame which the drone will attempt to fly directly above. Next you have to engage position control. You can engage this mode with the *p* key, and disengage with *v* for velocity control. So the procedure is to first save a frame (target location for position hold) using *r* and then shortly after (before drifting too much) type *p*.

Note: Position hold works best over a textured surface with lots of visual contrast. Even when doing position hold, always be ready to kill in case of a mishap. Especially be careful when looking at other windows.

3) Position Control Demo

This video demonstrates the drone doing a zero velocity hover and drifting in the scene. Then we turn on position hold (you can tell when it is engaged when the drone's throttle drops) and it holds its position for several minutes.

Then we turn off the position hold so you can see it drift again, and then turn it on again at the end and land. You can tell when it is turned on because we move the drone back to the center of the flight area before each hold.

3.9. Problem 1: Flying with Velocity Control

First, you are going to experiment with flying your drone in velocity control and controlling its motion with the keyboard keys. Based on observations and knowledge of the controllers, you will then explain the inner workings of the velocity PIDs in your own words.

Setup
Prepare your drone to fly over a highly textured planar surface. Make sure there is space for the drone to fly around.
Exercise
Fly your drone in velocity control (the default control) and make sure there is room to fly to the right. Press and hold 'L' and observe the drone's motion, and release 'L' to stop the drone from moving.

1. Explain what the following key terms are in this controller, and how they change to cause the drone to move when you press 'L' and stop when you release: setpoint, error, control variable, process variable, proportional term, integral term, derivative term. We are looking only for a higher level description to demonstrate understanding of the PID controllers.

2. Try flying in velocity mode over a blank white poster board. Be careful! What do you notice about the drone's behavior, and what do you suspect causes this?

3.10. Problem 2: Flying with Position Control

Now you are going to fly your drone in position control and experiment with controlling its motion with the keyboard keys. Based on observations and knowledge of the controllers, you will then explain the inner workings of the position PIDs in your own words.

Setup
Prepare your drone to fly over a highly textured planar surface. Make sure there is space for the drone to fly around.
Exercises
1. Engage position hold using the procedure described above. Observe the drone's behavior. How is it different from just velocity control?
2. How long are you able to hold position? Ideally you should be able to do this in one

spot for an entire battery. If not, try re-tuning your I-term preloads above. If you're flying on the power supply instead of a battery, the drone should stay in place indefinitely, but you can stop it after 5 minutes.
3. While flying in position control, make sure there is room for the drone to fly to the right and then take note of the desired position in `4 of the screen. Now press the 'L' key in the user interface and note the new desired x-position of the drone; it should be 0.1m to the right of the drone's last position. Explain what the following key terms are in the outer loop position controller, and how they change to cause the drone to move and stop 0.1m to the right after you press 'L' in position control: setpoint, error, control variable, process variable, proportional term, integral term, derivative term. We are looking only for a higher level description to demonstrate understanding of cascaded PID controllers.
4. Try flying in position control over a uniform surface such as the floor in 121, or unpatterned carpet. Echo the state of the drone by typing `rostopic echo /pidrone/state` into an empty window in the screen. Note the position data, and explain your observations of how well the drone is able to estimate its position. How long is it able to hold position? Does the drone move correctly when you use the arrow keys?

Take a one minute video of your drone flying in velocity control, and then engage position control.

Section L
Localization

This section introduces robot localization: the process of determining where a robot is in a pre-defined map. The content and instructions in this section will teach you how to create a map for your drone to fly over, and how to command your drone to fly to specific positions in your map. For the drone, a "map" is simply a photograph of the surface that your drone flies over.

Subsection L.1
Camera

This subsection describes how the downward-facing camera on your drone is used to localize above a map. By storing a picture of the flying surface on the drone and running the localization program, your drone will be able to determine its x and y coordinates within the map.

Unit L.1.1
Comparing Images

Image that you're starting to assemble a jigsaw puzzle. You take out the first piece and then you look at the picture on the box to guess where that piece belongs. If the puzzle piece has many differnet colors or distinct lines, then it is easier to find where it belongs in the whole picture. however, if the puzzle piece is a solid color, say a blue part of the sky, it is very difficult to narrow down exactly where the piece belongs. As you find more pieces that fit with your first piece, it becomes easier to determine where those pieces belong in the picture.

This is how localization on the drone works. The drone has a picture of the entire surface that it will fly over stored on it; this is called the *map*. Compare the map to the complete picture on the puzzle box. As the drone is flying, the camera is only able to see part of the map. The drone tries to find where this picture belongs in the map, just like finding where the puzzle piece goes. Once the drone knows approximately where the picture belongs, it can guess how far to the right, and how far up it is flying from the bottom left of the map. And viola, the drone has a position estimate!

Of course, there is a lot math and code behind the picture matching process of the drone; however, the higher level understanding is just as described.

1) Compiling Map Image For Drone Use

Fotor Image Compiler

1. Using the link above head to the Fotor image collage tab.
2. Choose Photo stitching to begin.
3. Set the spacing to 0 and check off the transparent borders box.
4. Lastly on the right hand side of the page click on the import button to upload the images of your map. (Make sure to have all pictures taken from the same distance to have the most accurate stitching).

Unit L.1.2
Drone Localization

The Drone flys over a known environment. To localize, state estimation is required on a system that changes over time.(The drone changes the view when flying over time.)

Guassian

Particle Filter

localization

2.1. Monte Carlo Localization

Monte Carlo Localization is localizing using particle filters.

Monte carlo Localization has 3 phases

prediction phase

Update phase

resamplying

2.2. Prediction Phase

During the Predicition phase the drone doesn't know where it is in the map and can equally be anywhere in the map.

2.3. Update Phase

During the Update phase the drone updates the particles to be aware of the where on the map it is at.

2.4. ReSamplying Phase

DUring the Resamplying Phase the weighted particles with higher numbers are recycled while deleting the lower weighted particles.

2.5. Particle FIlters

Particle filters represents a Probability Distrubution Function. A Probability distrubution function grabs the probability of a given event.With 50 features, the PDF will give a weighted probability of those 50 features.Given the random features on the camera, the Filter can give the probability of the location by weighting out each particle feature it has. The more weight the particle has the more likely that particle has to do with the location of the drone then a particle with a lower weight.The higher weight gets recyled in the filter while the lower weights are deleted. Deleting the lower weighted particles from algorithm, helps to effeciently localize because only the higher weighted particles are keeped in the algorithm while searching for more particles.

The map used for the DuckieSky Drone is distict around the map because the more features the better! The accuracy of particles is relatived to the ammount of features used to represent the Probability Distrubution Function.

How will the map work when the marks on the map are the same?

How do weight particles effect localization?

Section M
Materials

This section contains other supplementary materials for this textbook.

UNIT M.0.3 Glossary

This glossary contains an alphanumerically ordered list of terms utilized in this text book. Some words or phrases are linked to a page where you can learn more about the definition of the term.

- AI (Artificial Intelligence)
  - Artificial intelligence allows systems to gain the potential to accomplish tasks that usually requires the intelligence of humans or decision making skills.
- Basestation
  - A basestation is a laptop or desktop (ie. not a tablet) with the ability to connect to WiFi over a network and that has the ability to run/read python.
- Bash
  - Bash is a type of program language. It is utilized in many shells.
- Branch
  - A branch is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch allowing you to work freely without disrupting the "live" version. When you've made the changes you want to make, you can merge your branch back into the master branch to publish your changes.
- Bystander Effect:
  - The bystander effect describes a phenomem where the more people that are present, the less likely someone will help a victim during a situation. Be wary of this, make sure that if there is a dangerous situation, be cautious and aware, and take action to help those who need it.
- B4UFLY App
  - B4UFLY App is an app created by the FAA, helps recreational flyers to figure out where they can safely fly and if there are any restrictions in a location.
- Cloning
  - A clone is a copy of a repository that lives on your computer instead of on a website's server somewhere, or the act of making that copy. When you make a clone, you can edit the files in your preferred editor and use Git to keep track of your changes without having to be online. The repository you cloned is still connected to the remote version so that you can push your local changes to the remote to keep them synced when you're online.
- Closed source code
  - when the source code cannot be accessed by others, or it remains classified, only seen by those who are authorized to.
- Directory
  - A directory is another name for a folder.
- DL (Deep Learning)

  - Deep Learning is a portion of AI that replicates the human brain by processing data involved with object and speech recognition, and making decisions (Investopedia).

- DuckieSky
  - DuckieSky is a program within Duckietown lead by Professor Stefanie Tellex that develops and manages the MOOC-based drone curriculum in this book and that distributes drone kits to high schools throughout Rhode Island.
- Duckietown Foundation
  - The Duckietown Foundation is the non-profit foundation that develops and promotes the Duckietown project, which explores autonomous robot platforms. You can read more about Duckietown.
- Ethics
  - The term ethics originated from the word “ethos”, which is Greek for “way of living” (BBC). Ethics incorporates moral principles and values. It affects how we choose to live our lives, what we think is wrong and right, and what our responsibilities are (BBC).
- FAA (Federal Aviation Administration)
  - Federal Aviation Administration which is responsible for regulation of civil aviation: including airports, air traffic management, certification of people, certification of aircraft, and protection of US assets.
- Fork
  - A fork is a personal copy of another user’s repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original upstream repository. You can also open a pull request in the upstream repository and keep your fork synced with the latest changes since both repositories are still connected.
- Git
  - Git is an open-source program for tracking changes in text files. It was written by the author of the Linux operating system, and is the core technology that GitHub, the social and user interface, is built on top of.
- GitHub
  - GitHub is a Git repository hosting service, or an online datastructure that is a basis for storing and presenting these code projects.
- HTML
  - HTML [Hypertext Markup Language) is a programming language used to create webpages.
- Issue
  - An issue is a suggested improvement, task or question related to the repository. Issues can be created by anyone (for public repositories), and are moderated by repository collaborators.
- Kill Switch
  - The kill switch on the drone is the spacebar on the keyboard which immediately

disarms the drone after they have begun flying it. If something goes wrong during flight, press the spacebar.

- ML (Machine Learning)

  - Machine Learning is a portion of AI that lets systems learn and improve from their experiences without programming it into the system.

- Markdown

  - Markdown is a text-to-HTML conversion tool for web writers.

- Markduck

  - Markduck is a Markdown dialect that supports many Markdown features. It is the language that the majority of the Duckiesky High School Textbook [including this document) was written in.

- NTSB (National Transportation Safety Board)

  - US investigative body for vehicle/transportation accidents.

- Open source code:

  - when the source code can be accessible by the public.

- Open-sourced Project

  - An open-sourced project is a project where the code used to make a particular program or application is available to everyone.

- OSHA (Occupational Safety and Health Adminstration)
- Pull Request

  - A pull request is a proposed change to a repository submitted by a user and accepted or rejected by a repository's collaborators.

- Repository

  - A repository is the most basic element of GitHub. They're easiest to imagine as a project's folder. A repository contains all of the project files [including documentation), and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private.

- See and Avoid Concept

  - The see and avoid concept is the concept of taking action to avoid what you see. If you are able to see a ball coming at you, you can take action: catch, dodge, etc. In the Midair Collision, time was too short to be able to take any action.

- Shell

  - A shell is a programming language that takes input and gives the input to the computer and operating system to analyze and perform the task that the input asks for.

- SSH

  - SSH (Secure Shell) is a method that allows a user to remotely log in from one computer/device to another.

- Syntax

  - Syntax refers to the rules that specify the correct combined sequence of symbols that can be used to form a correctly structured program using a given programming language.

- Terminal
  - A terminal is a program that allows the user to interact with the shell.
- UAS (unmanned aerial systems)
  - systems/vehicles with no human pilot, like drones.
- Version Control Systems
  - Version control systems are software programs that allow programmers and code-based project workers to manage the changes to their code-based projects over time with new versions.
- VS Code
  - Visual Studio Code (VSCode) is a code editor offered by Microsoft.

SECTION N
Bibliography

Please see [1].

[1] Roger R Labbe Jr. Kalman and bayesian filters in python. Published as a Jupyter Notebook hosted on GitHub at https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python and in PDF form at https://drive.google.com/file/d/0By_SW19c1BfhSVFzNHc0SjduNzg/view?usp=sharing (accessed August 29, 2018), May 2018.