Engineering Physics Appendix

Kevin Robb
kevin@js-x.com

Capstone 2020-2021
May 12, 2021

Contents

A Engineering Physics: Design Process **2**
A.1 Define a Problem . . . 2
A.2 Brainstorm . . . 2
A.3 Research . . . 2
A.4 Identify Criteria and Specify Constraints . . . 3
A.5 Explore Possibilities . . . 5
A.6 Select an Approach . . . 5
A.7 Develop a Design Proposal . . . 6
A.8 Make a Model or Prototype . . . 6
A.9 Test/Evaluate Design . . . 6
A.10 Refine the Design . . . 9
A.11 Create or Make Solution . . . 9
A.12 Communicate Processes and Results . . . 9

B Engineering Physics: Eight Considerations **10**
B.1 Economic . . . 10
B.2 Environmental . . . 10
B.3 Sustainability . . . 10
B.4 Manufacturability . . . 10
B.5 Ethical . . . 10
B.6 Health and Safety . . . 10
B.7 Social . . . 11
B.8 Political . . . 11

A Engineering Physics: Design Process

What follows are the steps in an engineering design process (according to ITEEA) and a description of how my project can be viewed within this framework.

A.1 Define a Problem

The main problem I am attempting to solve with this project is that of mobile robot perception. It is very important for a robot to acknowledge, measure, and understand its environment, especially if that information is immediately necessary for its autonomous behavior. Methods such as gmapping and SLAM are moderately successful at creating a rudimentary map to serve as the robot's internal world model, but the robot still has the trouble of placing itself on the map and updating it appropriately. A solution to this problem – a perception system which can effectively utilize all available sensors to accurately track important system states – would be a great boon for anything from search-and-rescue drones to scientific mission rovers.

A.2 Brainstorm

Rather than tackling SLAM or another fairly complicated mapping technique, we will focus on the robot's own sense of position (and other states such as velocity and heading). Ideas for addressing the problem in this way involve Kalman Filters, a sensor-fusion method which seems to fit our situation, as well as Machine Learning. ML can provide the dynamicism we seek as well as propel training along automatically, rather than us needing to manually tune the KF and evaluate it for long periods of time. My proposed project is to implement my own custom Kalman Filter which takes in data from GPS, IMU, velocity readings, LiDAR, and the bumper, and keep live estimations of the position and velocity in each coordinate. I would like to include the heading (yaw) and the yaw rate, but this will require an Extended (non-linear) Kalman Filter, which may be too complicated for me to implement. Thus the initial proposed state is:

$$\boldsymbol{X} = \begin{pmatrix} x & y & \dot{x} & \dot{y} & \theta & \dot{\theta} \end{pmatrix}^T \tag{1}$$

A.3 Research

For the Kalman Filter implementation, my primary resource is kalmanfilter.net, which gives a fairly good and thorough introduction to the 1D linear Kalman Filter, as well as some preliminary derivations for a simple multidimensional KF. I will be creating my own summary document as I go through it, to ensure my understanding of the equations and how the different aspects of the filter work in tandem, and I will also use this summary document to track my progress through different sources, cataloguing and merging them into a useful body of knowledge to reference while implementing it in code,

In terms of the neural network, a resource that I have been looking into is the Neural Networks and Deep Learning textbook, which gives a good introduction to perceptrons and the basics of neural networks. I will also be referencing Reinforcement Learning: An Introduction, the classic work on RL by Sutton and Barto. At this point I thought it was more likely I'd pursue a supervised learning approach (such as a neural network), but I ended up switching over to a more applicable evolutionary computation approach that falls closer to the realm of reinforcement learning. I will keep in mind Machine Learning, Tom Mitchell's 1997 book, for any conceptual questions.

Aside from textbooks, there are a number of papers out there with similar problem domains which I have used to gain insight into this field of robotics and perception. Some papers I have obtained are:

- *Adaptive Unscented Kalman Filter for Target Tracking with Unknown Time-Varying Noise Covariance*, Ge et al.
- *Analysis of a federal Kalman filter-based tracking loop for GPS signals*, Jin et al.
- *A Robot Architecture for Outdoor Competitions*, Oliveira et al.
- *Autonomous State Estimation and Mapping in Unknown Environments With Onboard Stereo Camera for Micro Aerial Vehicles*, Sun et al.
- *Design of an Autonomous Racecar: Perception, State Estimation and System Integration*, Valls et al.
- *Distributed Kalman Filter with a Gaussian Process for Machine Learning*, Jacobs and DeLaurentis
- *Drift Calibration Using Constrained Extreme Learning Machine and Kalman Filter in Clustered Wireless Sensor Networks*, Wu and Li
- *A Novel Fifth-Degree Cubature Kalman Filter for Real-Time Orbit Determination by Radar*, Li et al.
- *High-gain extended Kalman filter for continuous-discrete systems with asynchronous measurements*, Feddaoui et al.
- *Improving Adaptive Kalman Filter in GPS/SDINS Integration with Neural Network*, Wang et al.
- *Integrating Extreme Learning Machine with Kalman Filter to Bridge GPS Outages*, Jingsen et al.
- *Model-Based Robust Pose Estimation for a Multi-Segment, Programmable Bevel-Tip Steerable Needle*, Favaro et al.

Many of these papers discuss specific applications of some things similar to my goals, but for different applications, such as advanced radar tracking of aircraft. I will be using these references to gain insight into the process of integrating a KF to a specific situation and deriving aspects such as the state transition matrix for a particular dynamic model; I will use this knowledge to derive the KF equations for my particular case, and to then integrate machine learning into the tuning mechanism.

I will be implementing the Kalman Filter myself from scratch, but for the machine learning component, I may use a publicly available package such as TensorFlow or scikit-learn to demonstrate functionality, and create my own implementation if time allows.

A.4 Identify Criteria and Specify Constraints

I will be focusing mainly on obtaining functionality in simulation, and moving to a physical robot only if time allows. I'm using a simulator created by a teammate in my student organization, Sooner Competitive Robotics, that we use to test our code for our competition robots. I have obtained permission to use it for my purposes, and the creator was kind enough to add a couple additional variable publishers so that I can access the ground truth for my state variables and more confidently tune the filter.

This simulator has different modes, which vary the number of obstacles as well as the sensor noise. The environment involves four random GPS waypoints which must be visited in order,

FIG. 1: A screenshot of a run in progress in the simulator. The red shapes are obstacles, and the yellow cone is one of the waypoints. The robot uses ackermann steering, much like a modern car.

and obstacles spread randomly across the map. I have created a rough agent which simply uses a PID (proportional-integral-derivative controller) to pursue the heading directly to the next waypoint from its current position; the obstacle avoidance is purely reactive, performing some pre-programmed avoidance maneuvers when the LiDAR senses a very close obstacle or the bumper is triggered. This agent is able to perform alright on the lowest noise and obstacle settings, but ramping up to realistic noise or increasing obstacle density causes it to get stuck or confused very quickly.

I will consider this project a success if my KF agent is able to complete the objective with at least 90% consistency on the realistic noise and average obstacle settings, and will be extremely satisfied to achieve the same consistency on the highest obstacle density mode. Through the course of implementing this project, my goals changed somewhat from obtaining peak performance in this simulated contest to maximizing the efficacy of the Kalman Filter itself. To this end, I defined a cost metric based on deviance from true position which is used to evaluate the success of any given run. The current hypotheses and goals for the project focus on getting the best perception and localization possible for this situation.

In terms of constraints, this is a two semester capstone project. By the end of the first semester (Fall 2020), my goal was to have the KF derived and integrated with my intelligent agent, which I achieved. I had also begun implementing the neural network, which by the start of the Spring semester I had decided would not reasonably suit my needs. My goals for the end of the second semester (Spring 2021) were to have optimized the KF with machine learning and be testing the concept with a physical robot. My goals changed considerably, and I decided to focus wholly on the software aspect to get as much out of the project as possible. The new goals for the Spring semester were to have demonstrated that the KF localization can be improved automatically via evolutionary tuning, and to attempt a similar approach to evolve a simple yet efficient motion and control policy. This latter piece turned out to be very tricky, and was not successful when done alongside the perception evolution; it would need to be more of its own project if we hope for a chance of success.

A.5 Explore Possibilities

Based on further knowledge, I extended my schedule to allow for more time with the Kalman Filter implementation, since it is far more complicated than I had originally believed. Particularly, I was under the impression that I could use three separate linear Kalman Filters to keep track of $(x, \dot{x})$, $(y, \dot{y})$, and $(\phi, \dot{\phi})$. This implementation turned out to be terrible, in part because I had underestimated how to handle the process noise and covariance, but I suspect mainly due to the correlation of these variables. Especially with the heading, ϕ, this really calls for an Extended KF, so I may need to devote even more time to that implementation, or only track the position and velocity system states, suffering from an increased model uncertainty due to the inaccuracies. I decided to leave out the heading from the state, and use a constant-velocity, constant-heading model, which ended up working out fairly well.

Due to the KF taking longer than expected, I now anticipate that I will be using a pre-made implementation of the machine learning method I choose to use, since implementing a neural network myself would be a good and interesting learning experience, but not as relevant to the goals of this project as the KF itself.

The KF took longer to implement than expected, so I planned to use an open source library for my machine learning approach; however, I found the time to implement my own neural network from scratch over the winter break. Unfortunately, as I've mentioned, this turned out to not be the best type of machine learning to use for this task, so I changed gears to an evolutionary approach. My research project in 2018 centered around evolutionary computation, and I'm very familiar with how it works, so I decided to implement it from scratch with my project as well. This went very well, but did mean that I spent more time on the implementation itself and less time on analysis and further iteration of the concepts as a whole.

A.6 Select an Approach

Lingering questions that persisted for a long time:

- What to use for the state?
  Only consider the position and velocities.

$$\boldsymbol{X} = \begin{pmatrix} x & y & \dot{x} & \dot{y} \end{pmatrix}^T \tag{2}$$

- How to include machine learning?
  Use TensorFlow, Keras, or scikit-learn's packages for initial implementation and testing. I really wanted to make my own implementation, which led to the neural network. When I realized this was not going to be best, I wrote an evolutionary computation program to interface with the rest of my code.

- What goal to focus on?
  I wanted to maximize performance in the simulator for a long time, but ultimately switched to attempt optimizing just the perception, which was far more successful as we could much more directly map our settings to the results, with less randomness getting in the way.

Final idea:
Implement a 4D Kalman Filter with my simple agent to improve performance in noisy environments, and tune its hyperparameters using evolutionary computation to optimize localization.

A.7 Develop a Design Proposal

This is a software project. I have discussed the simulator I will be using. This runs in Robot Operating System (ROS) Melodic, with Ubuntu 18.04. Languages available to me are C++ or Python; I attempted an initial implementation in C++, but experienced a lot of issues, so I will be using Python. I am writing data to a series of CSV files, which I then analyze with some custom R scripts. My GitHub repository is available here.

In the summary document in the repository, I am adding equations, derivations, and explanations as I come across them which will be useful for my own or others' understanding, and are mainly intended as reference for myself as I am coding. It is an R Markdown document which I knit to HTML after each section is complete; it is available here. The specifics of the Kalman Filter which I wrote after condensing all the scattered information in my summary document are included in Appendix **??**.

A.8 Make a Model or Prototype

The master branch of my GitHub repository contains my working prototypes at any given time. I first created a working agent that has no KF, ML, or really much intelligence whatsoever. I then implemented my first topic of discussion, a 6D KF that did not use matrices and performed just as if it were three constant-velocity 1D models. I then created the body of plotting software to make visual evaluation of all runs of the simulation very straightforward. After using what I learned from the plots to fix some errors in my simple agents and control code, I converted the KF to an entirely matrix-based implementation with a 4D state, $(x, y, \dot{x}, \dot{y})^T$. After tuning the KF and vastly improving my plotting scripts, I implemented the evolutionary computation driver and agent files. I initially used my manually tuned KF parameters for the initial genome, but later changed it to be randomized, and also went through several different versions of the genome. Ultimately, isolating it to only perception parameters was the most successful.

A.9 Test/Evaluate Design

The initial 6D model I discussed was only able to perform with an estimate uncertainty so close to 1 that the agent may as well be going entirely off of its measurements (as it did before the model was implemented at all!). With more confidence in its estimations (which rapidly fall away from the truth as the robot changes its heading in the course of pursuing its path), the robot becomes completely lost and eventually hits obstacles enough times to "die," ending the simulation. This is considered a pretty stark failure.

I was stuck for a while on the new matrix-based 4D KF, which almost immediately blew up to the max values for x and y, crashing the filter. After a lot of manual tweaking, mainly consisting of lowering $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$, it became able to find the ground truth fairly quickly and accurately track changes in the State. Small changes in these values have a large effect; FIG. 3 shows how the State lags behind the truth if its covariance is too high, and FIG. 5 shows it being much more successful after more tweaking.

One point of note is that the covariance matrix, $\boldsymbol{P}$, is in a constant state of increase and decrease. If the filter has found the ground truth, and the truth is continuing fairly regularly on its predictable trajectory, then the KF converges on the truth, decreasing $\boldsymbol{P}$ close to zero. When the State experiences rapid changes, such as when the robot arrives at a waypoint and suddenly turns to pursue the next one, the KF falls off the truth as it tries to keep up with the change; $\boldsymbol{P}$ increases to reflect the higher uncertainty in the internal state and the higher reliance on measurements to find the ground truth again. This process can be seen with a close look at x and y in FIG. 5.

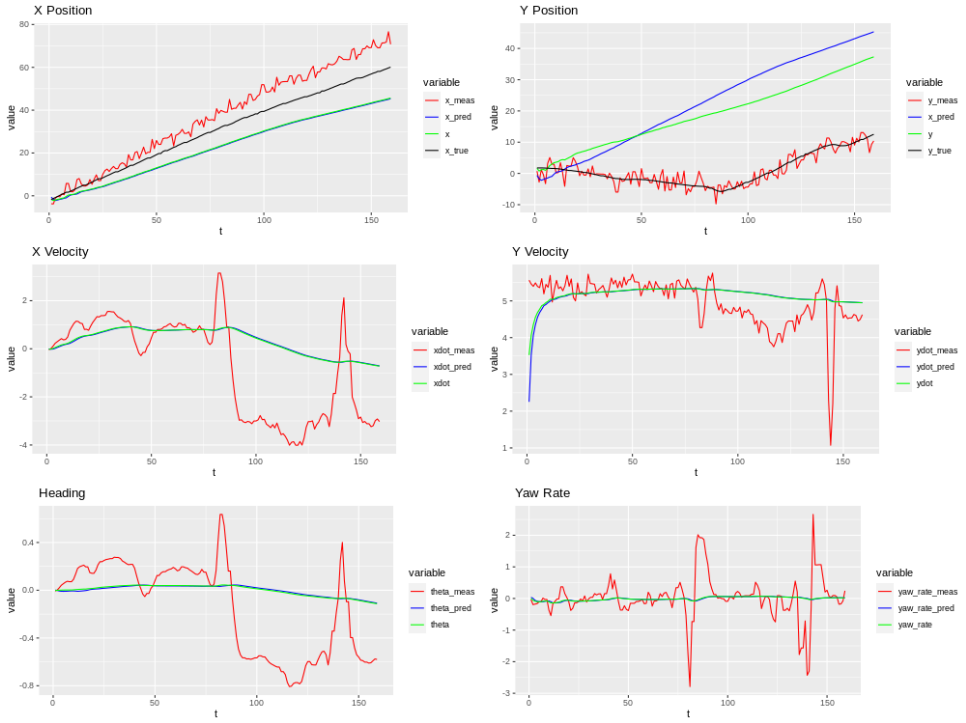

FIG. 2: The uncorrelated 6D Kalman Filter's performance. (It's pretty bad.)

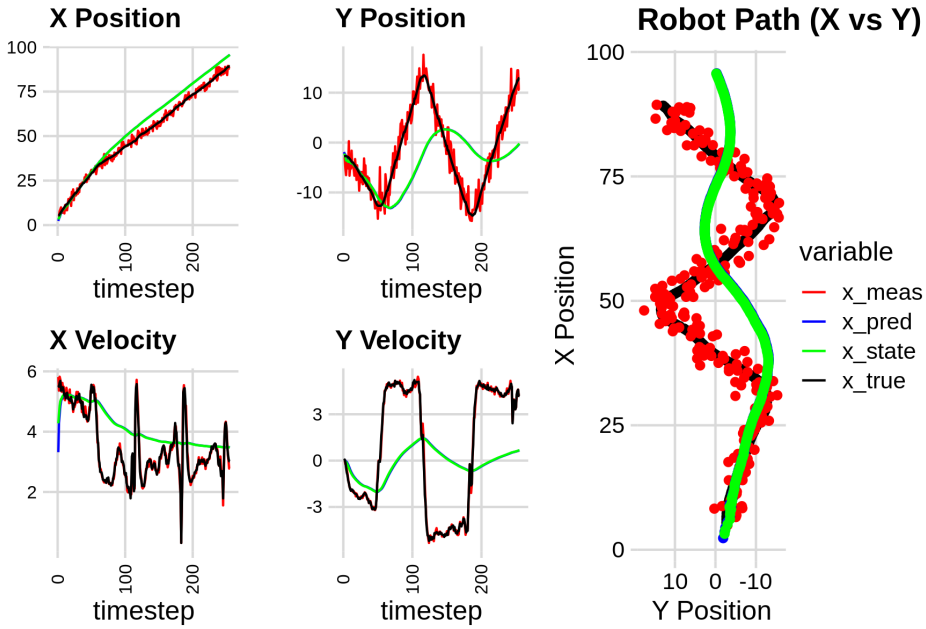

FIG. 3: The 4D KF lagging behind the truth because its process noise is too low, so it believes its predictions are better than the measurements, and doesn't adhere to their trend as much as it should.

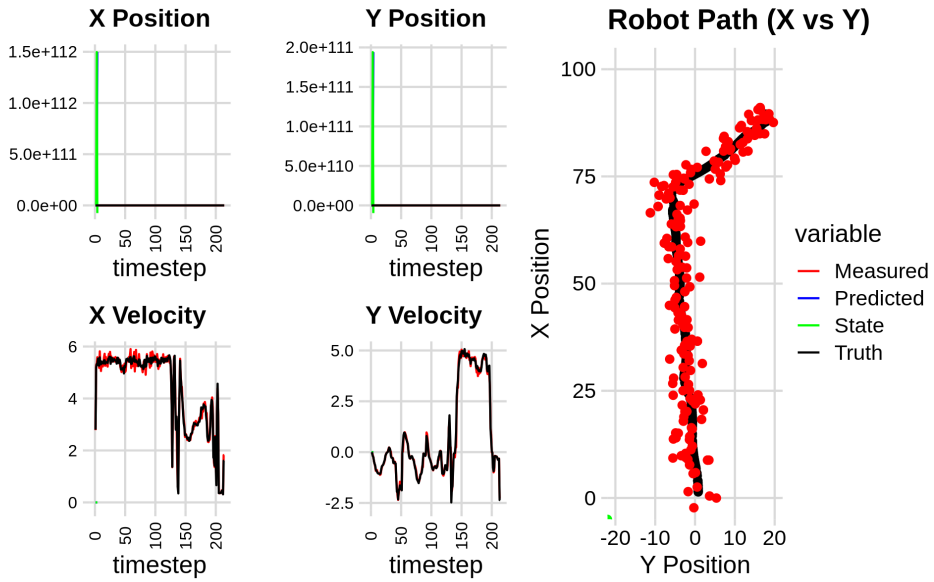

FIG. 4: The 4D KF blows up to inf/nan because the covariance is too high for it to get enough measurements and correct it, and the noise prevented the Kalman Gain from dropping below 1. Note the scale on x and y, and the point for the state at the lower-left of the track.

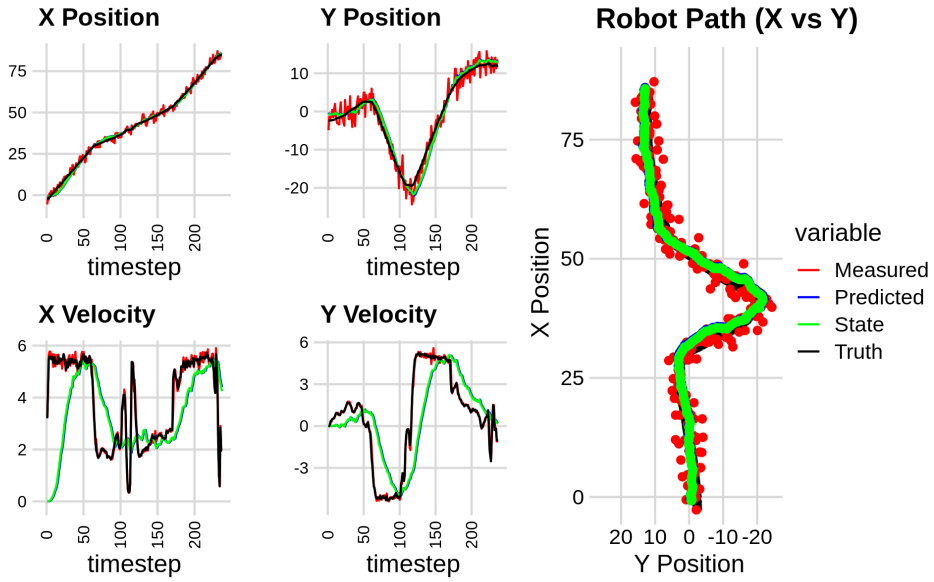

FIG. 5: The 4D KF performing well. Note that it is quick but not immediate to react to changes in the State variables.

Another point worth noticing is that the filter approximates x and y very well, but is not as good with $\dot{x}$ and $\dot{y}$. This is because we are using a 4D state and a constant velocity model. This means we are assuming the velocity is constant (even though it clearly isn't) just to give the KF somewhere to start. We need to assume something is constant in order to create our dynamic model in the first place, and this must be one order higher than our desired precision. So, if we want to get accurate estimates of not only position, but also velocity, then we would need a 6D KF with a constant acceleration model. We are concerned in this project primarily with the position, so this is fine for our purposes.

A.10 Refine the Design

I spent the majority of the Fall semester overhauling the KF to induce correlation between the variables, actively track and update covariance, and tweak the uncertainties of both the measurements and estimates to provide the best system state predictions. Switching the system from separate linear KFs to one matrix-based KF was a step in the right direction, and now that it's been manually tweaked, it works fairly well in its own right.

Adding the machine learning component introduced a slew of new problems and poor performance, but once it was working, the KF can now be tuned automatically from an initially random genome to achieve the same average performance as the meticulous manual tuning produced.

A.11 Create or Make Solution

This project is purely software based, so there is no need to consider mass production or packaging. This is an interesting concept that I intend to pursue further in my graduate studies and potentially in my career. I'm interested in robot perception and autonomous motion planning, both of which this project attempts to investigate and improve upon.

A.12 Communicate Processes and Results

This project will have been presented with other capstone seminars three times by the end of the 2020-2021 academic year. It will also have been presented as many times to the research group in which I am working, Dr. Dean Hougen's Robotics, Evolution, Adaptation, and Learning (REAL) Lab. Much of the technical aspects are presented within this document.

Potential future work includes applying the same ideas to evolving motion and control parameters in addition to perception. It would be interesting to see how the agents learn to avoid obstacles and recover from collisions, but that was outside the scope of this project. It would also be a project in its own right to attempt a similar concept with a physical robot in the real world, although that would likely require a custom simulator designed to be as near the real thing as possible.

B Engineering Physics: Eight Considerations

B.1 Economic

Improving robot perception not only improves safety and benefits humans in the same environment as a robot, but it also helps the robot to more quickly and effectively complete a task. This can include better obstacle avoidance to prevent damage to the robot, more robust path planning to find the best path to a goal location, or faster reactive behavior like shutting down to prevent human injury. This more complete and regularly updated world model that I have designed in my project is a way to use all available sensor information to collect and process as much information as possible, maximizing efficiency. This improvement to efficiency makes a difference in a commercial application, even at small scale in a single warehouse.

B.2 Environmental

This is possibly the least applicable of the eight considerations. A point of notice is that increased performance of drones and autonomous vehicles will reduce damage to nature, such as by preventing a robot from losing its location and being unrecoverable, left in the environment.

B.3 Sustainability

Sensor fusion algorithms such as my project are a hot topic for research, and are only improving each year. This research is generally in the public domain, with the goal of everyone helping each other to progress our field as rapidly as possible to its highest potential. This research helps keep this trend going into the future.

This specific field of advancing perception also helps sustain the field of robotics and autonomous vehicles in general, as it makes them more suitable to an expanding set of applications, and more likely to be accepted by the public and lawmakers.

B.4 Manufacturability

The purpose of this algorithm is to use all available sensor values together in order to maximize knowledge of the environment and the accuracy of its estimations. As such, this software, or a similar structure, can be implemented on any pre-existing system and made to use the sensors already available; if more or better sensors are obtained, they can easily be installed or upgraded.

B.5 Ethical

The main ethical benefit of this project is the progression of robot knowledge, and thus the increase of safety. This is expounded in the next subsection.

B.6 Health and Safety

Autonomous robots are being increasingly used in both the commercial sector and the service industry, and these robots will inevitably be around or interacting with humans. This includes automatic warehouse transports, self-driving cars, and grocery store floor-cleaners; all of these applications benefit greatly from increased perception, as it allows them to more accurately perceive the world and avoid danger to humans. Knowing more about the environment benefits the robot in that it is better able to complete its task, and it benefits humans in that we will

be more safe in our use of these autonomous machines and our eventual cohabitation of the same spaces.

B.7 Social

An important aspect of robotics is robot-human interaction. Robot use in the service industry is furthering these relationships, and we can ensure this relationship remains positive by increasing knowledge and safety. People are wary of autonomous applications such as self-driving cars, so progress in the field of perception (i.e., my project) will help to assure the public of the safety and redundancy built into these systems.

B.8 Political

Similarly to the general public, politicians are wary of new technology that they don't understand. Generally, government regulations lag behind technological advancements, so we can get ahead of the curve in a sense; by maximizing the extent to which a robot or autonomous vehicle can see, understand, and react to the real world, we improve the likelihood of positive relationships and fair regulation, and hope to prevent something like a ban on these technologies in the public sphere which would only hinder progress. By taking safety and accurate perception seriously from the get-go, we make sure that preventative measures are not necessary.