# Vehicle AI

### AItomotives

|  |  |  |
| --- | --- | --- |
| Brandon Goodhue | Hunter Kelley | Christopher Salvati |
| Stephen Cosco | Trevor Lezynski |  |

### Project Sponsor

### Daniel Krutz

### Faculty Coach

Pavan Pigudu

### Project Overview

The central conceit of this project was to develop a system to allow for dynamic and autonomous path planning for unmanned aerial vehicles (UAVs). The team was expected to do this using a pre-designed evolutionary Recurrent Neural Network (eRNN), called Evolutionary Exploration of Augmenting Convolutional Topologies, or EXACT for short. The specifics of how said system worked were not important for the team to understand, merely how to interface with it so predictions could be generated for path planning. Several pieces of off-shelf-software were required, namely Ardupilot, a drone simulation software, ROS, a system designed for running code on drones, and Mission Planner, a kind of virtualized control station for drones. In essence, the team's primary job was to generate the connective code to get these four components working together to facilitate a system for drones to be able to pick their own paths given a list of waypoints to travel to.

### Basic Requirements

The project was intended to increase the resiliency of autonomous path planning processes through integrating a novel evolutionary recurrent neural network(eRNN)-based approach to account for volatility with unmanned aerial vehicles. Our goal was to integrate the eRNN into a path planning process and evaluate our concocted process against existing machine learning processes. It was intended for these simulations to take place in both a simulated and physical environment.

### Constraints

There were very few constraints specifically from the sponsor. The sponsor provided a specific machine learning algorithm “EXACT” that we were to use to help us implement the system. Other than integrating this one algorithm and using it as a black box, the rest of the technical requirements were up to the team. The team took the first few weeks searching for tools that we thought would work well to help us with the problem, and then discussing the benefits and drawbacks with our sponsor. The main thing that constrained us was our huge lack of domain knowledge. The team had no prior knowledge of working with AI or machine learning, doing research, and very little experience with drones of any kind. This meant that we spent a lot of time in the beginning just trying to understand the problem space and understanding what kind of tools we would be able to leverage.

Something that was more of a process constraint was that this project was more of a research project than a standard software development project. This meant that the team had to adjust our process to a type of work that was new to us. Some of the difficulties that came with this change were in estimation and available information. Some of the tasks we had were things like gathering information or learning about specific technologies. Stories like this are inherently hard to estimate so the team had to be careful when dedicating time and giving our sponsor estimates. On top of this we had to make estimations for future work based on information we were still learning, so anything in the future felt impossible to estimate. In terms of available information, while working on a problem that hasn't really been tackled before, it becomes difficult to search for.

Overall the lack of constraint was both helpful and hurtful to the team. The team just was given a broad goal of designing an algorithm for a drone and given the entire internet to figure it out. Having no constraints allowed us to tackle this problem in whatever way we thought would be best. However this unlimited freedom in an area that none of us were experts in led us to making some poor choices that we eventually regretted. Possibly a little more constraint could have been helpful in giving the project a little more direction.

### Development Process

The team looked at this project mostly as a research project and not a typical system design project. This led us to not use the standard flavor agile scrum as we thought that we wanted more flexibility. We reasoned that with research, we would be learning new things too frequently to say exactly what we could have done in a week, and have a week or two planned out ahead of us. However since most of the team was still most familiar and comfortable with agile, we decided that a kanban style board would be beneficial to us. This way we could just have a constantly evolving backlog where the highest priority stories could always be picked up. This gave us a lot of the flexibility that we needed with still the benefits of agile. This included for us having virtual standups in a discord server and longer meetings twice a week, one of those being with our sponsor. For longer term that our kanban board we had a long term low precision google doc for a deliverables schedule. This was to have an expectation and keep a plan for far out deliverables, but was meant to be fluid and change as we learned new things.

The sponsor did approve of our process choice and the reasoning behind it, but did not mandate anything or attempt to influence how we managed our team. He was very willing to work with us and whatever process that we thought would be best. This included making time to meet with us as frequently as possible. He made himself available via email at all times and agreed to meeting once a week unless meeting more often was needed. Every meeting we would look at the deliverables document with our sponsor and update it according to what we did and what we learned throughout the prior week. This allowed us to have as much of a shared vision with our sponsor as possible, and all expectations were in writing. Since this was a meeting and not just a contract or an email these expectations were also completely able to be discussed and changed every week.

Our team did not identify many different roles for people, but Trevor was determined to be our communication lead. He was mostly in charge of all email communication to make it easier for our coach and sponsor to have a single point of contact.

### Project Schedule: Planned and Actual

When the team initially became acquainted with the project, the first thing we noted was the mass amount of uncertainty it entailed. None of us had any experience with machine learning, research, or any of the topics involved. We realized a good amount of our initial project work would be domain analysis and acquainting ourselves with the tools we were expected to use. Even disregarding EXACT and all actual machine learning systems, we had to learn how to use technologies like Ardupilot and Mission Planner, which is what took up the majority of our first semester.

From the outset, we decided that our principal goals for the first semester would be to get acquainted with how our simulation software worked and to set up a test bed for us to deploy code into. This would involve being able to control the drone through scripts rather than the U.I for Mission Planner, pull out data from the drone mid flight about various statistics, such as wind speed, position, and the like. The hope was that, in designing this system, it would make integrating with EXACT as easy as possible.

In practice, designing this testbed was a lot harder than any of us imagined. Firstly, out of the box mission planner was not designed to allow for non-linear commands to be issued. This means that Mission Planner expects commands to be a sequence of instructions to be executed without any logic involved, while we needed commands to be executed based on the changing state of the drone and the waypoints that remained to be visited. Initially, the team tried to come make our own modifications to Ardupilot and Mission Planner to make non-linear command execution possible, but this turned out to be a much more difficult task than was what was necessitated.

Shortly after the team started work on breaking linearity, we found ROS, a system designed to extend the functionality of Ardupilot that would allow us to dynamically issue commands to a local server, and have that server direct the drone. This was discovered the week before winter break began, and so our work over break was split between breaking linearity the way we initially intended, and investigating ROS as a development option.

When the team came back together after winter break, we decided to form a proper schedule, after completing the initial goals of setting up docker containerization and switching to ROS for the baseline of the system infrastructure. The resulting schedule was as follows:

|  |  |
| --- | --- |
| **Deliverable Date** | **Items to Deliver** |
| 2/18/21 | Deliverable plan, Able to build and run ERNN. |
| 2/25/21 | Establish a utility function as well as some training data.  Design a model for how the ros node is going delegate decisions. |
| 3/4/21 | Begin implementing the architectural design for the ros node. |
| 3/11/21 | Implement the architectural design for the ros node.  Midterm peer evaluations.  Coordinate final presentation attendance. |
| 3/18/21 | Deliver the implementation of the ros node for delegating decisions to the drone.  How to make a poster presentation attended.  Mid term meeting with coach. |
| 3/25/21 |  |
| 4/1/21 |  |
| 4/8/21 | Project poster concept. |
| 4/15/21 | Preliminary project poster.  Draft of final presentation. |
| 4/22/21 | Give/attend final presentations.  Create a video of what the results of our project were.  Deliver final poster. |
| 4/29/21 | Schedule final reflection meeting.  Attend presentations.  Poster presentation during SE Project Day. |
| 5/6/21 | Attend final reflection meeting, Draft technical report, individual peer evals, final project artifacts, team final self assessment, final technical report, summary of project reflection meeting, project website and repository up-to-date, course evaluations, summary of final reflection meeting, senior survey. |

Important to note is that we intentionally did not schedule anything for the weeks of 3/25 and 4/1. This was to give us two "floating weeks" in case work fell behind schedule, and these could be used to push planned deliverables back in case it became necessary.

The biggest problem we ran into was actually getting our system to be able to make decisions. While we figured out how to interact with exact and use it to make predictions, the principle issue came in the form of figuring out how to get our hands on a dataset that could be used to to actually make those predictions. Initially our sponsor provided us with the BlackBird dataset, a massive repository of data created by an MIT team that tracked various statistics across terabytes of drone flights. While this was alluring, it ended up being a massive time sink trying to make it work for us.

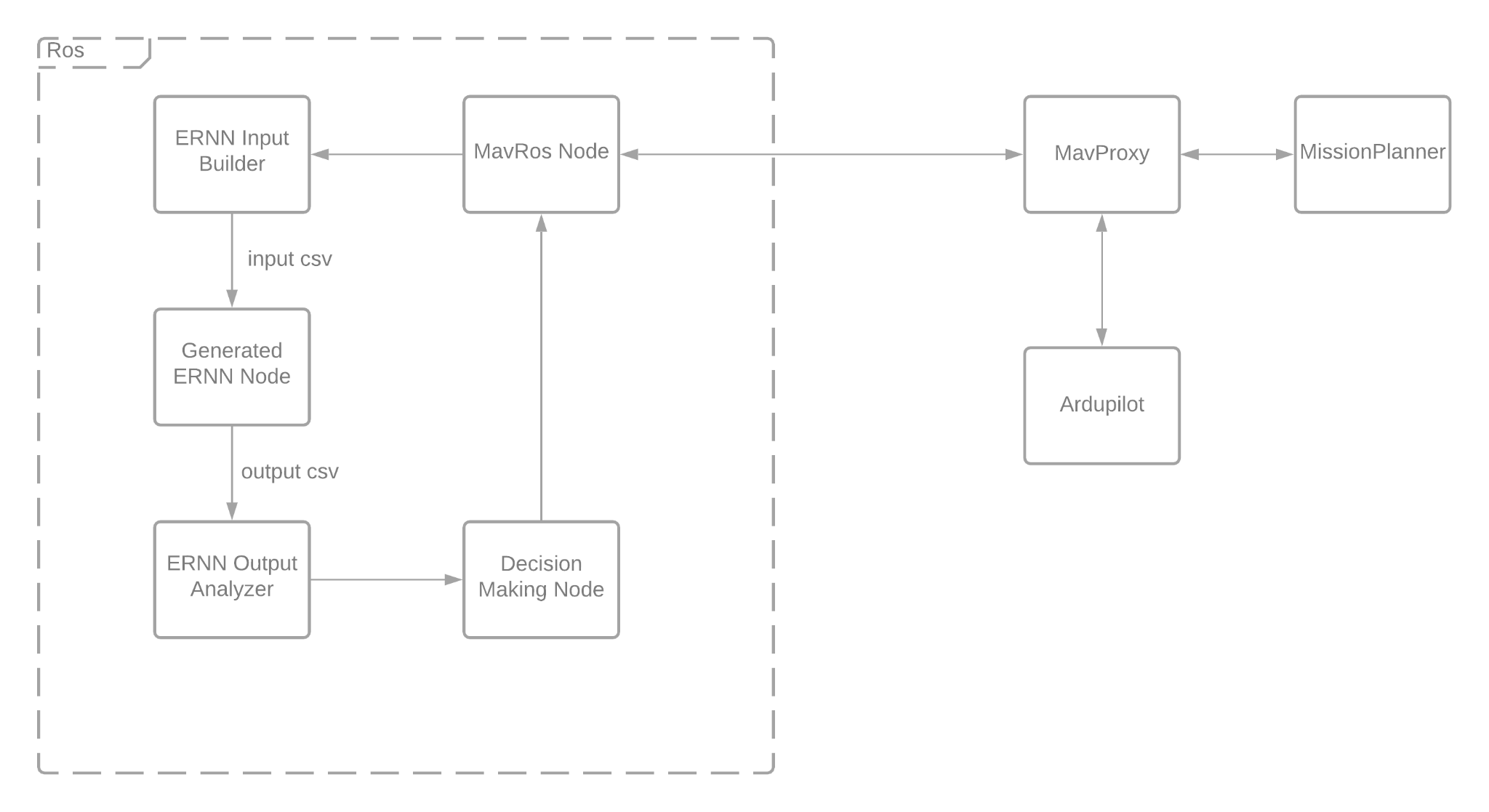
Unfortunately, to the best of what we could ascertain, the project was abandoned, and not very well documented in terms of how to actually access it. Due to the size of the dataset, they designed tools to allow people to pull down specific sections of it, and to the best of our ability, we could neither consistently build the tools, nor read any data we managed to pull. The team had used up the allocated week trying to get this to work, along with both of our floating weeks.

Eventually, in the interest of moving forward, we found a secondary, easier to use dataset, published by CMU, however, this was a much smaller dataset, that didn't have everything we wanted. The principle issue is that, while this dataset kept track of things like the battery voltage and the battery current, it didn't directly keep track of the battery percentage. This would cause a lot of friction and slowdown throughout the rest of the project.

As the team went on to try to use this dataset, we ran into major issues due to this fact. Mainly, we had thought that we could infer the remaining battery percentage from Arudpilot by keeping track of the battery voltage. In the case of a real battery, as it reaches zero percent charge, the voltage drops along a predefined curve based on the specifics of the type of battery being used. Ardupilot, however, doesn't do this. To the best of the team's inspection, the voltage of the simulated instead fluctuates the voltage randomly over the course of the flight, making prediction impossible.

By the time the team had realized this issue, it had gotten too late, and there wasn't sufficient time to either look for a new dataset or come up with an alternative solution. After talking with the sponsor about these newfound problems, the team decided that, instead of actually doing path planning, we would ensure that the software we delivered would be something that could be built into something that does autonomous path planning.

### System Design



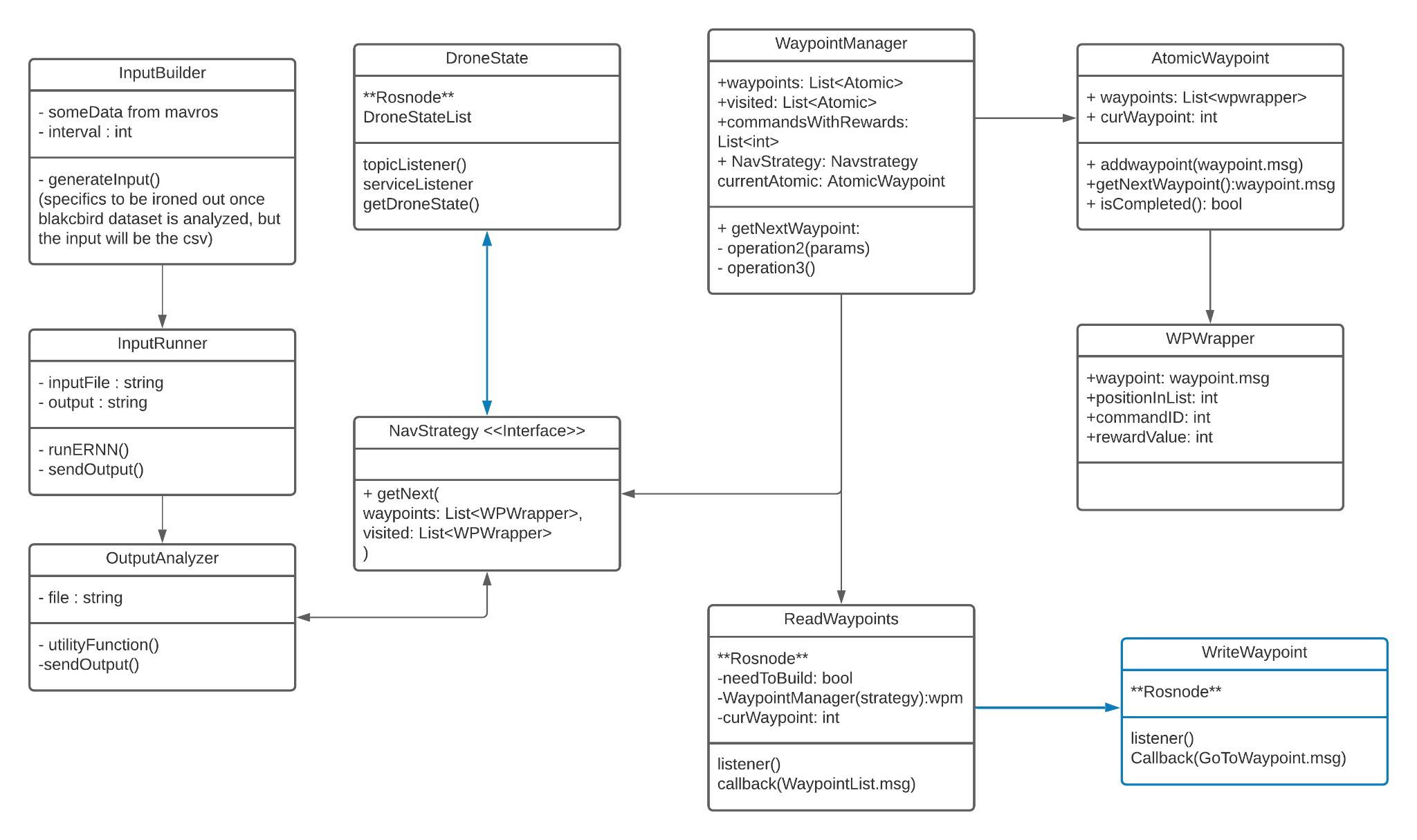
Mission Planner - A ground control software that is able to to create missions for the drone as well as interact with some drone parameters. Their parameters include things that allow for variance including wind speed and direction and battery percentage. This software is like a front end for the user to see what the simulated drone is doing and information about it in real time. Our system uses this as the place to feed a drone test scenarios and is also where we have a script to edit these parameters.

Ardupilot - An open source drone software that handles a lot of the actual drone piloting and simulating. Everything but the brain of the operation. Center point of all communication between the tools.

Mavlink - This is the communication protocol that drone commands and messages are sent with. This is the method through which all of our different tools and components are able to communicate with each other. We had to modify this protocol to be able to send and receive different types of messages, as well as modify other parts of our system to utilize the customizations we added.

ROS - Robot operating system is a node based publisher/subscriber framework. We are able to have multiple nodes or programs running that can communicate with each other in a pubsub fashion and through a library communicate with ardupilot. This communication goes both ways as our nodes read things such as the drones currently programmed mission and current state, and it can write back to the drone what decisions it should be making. Within our ROS system is where most of our effort was directed and it's the majority of our system.

Since this was the majority of our system this is also where most of our design decisions were made. Within the ROS architecture there are three ROS nodes of our own creation, each one having its own purpose, and then a library called mavros. This library has multiple nodes and services that act similarly to an api for ardupilot. Because of the pub/sub nature of ROS architecture the team’s nodes are able to subscribe to the nodes of the library and receive data from the drone. The team’s 3 nodes have different purposes and interact with different parts of mavros.

****

One of the nodes is responsible for writing to the drone a waypoint that it should be going to next. The team decided that this should be a separate node so that it has the potential to subscribe to multiple different sources. This node being separate also gives it the single responsibility of having to talk to mavros, without being coupled to any logic.

Another node is responsible for reading and remembering all of the drone’s state. This node has slightly more responsibilities as it reads from multiple different mavros nodes and compiles information. Since the neural network needs data from more than just one instant, this node also keeps track of a number of the most recent readings from the drone.

Not every mavros node updates at the same rate. For instance the position of the drone is updated 30 times every second, while the drone’s battery status is updated twice per second. Since this node is tasked with reading from multiple nodes and compiling them together, it works on its own time. Within the node there is a dictionary that is constantly updated with the most recently published data on all of the mavros topics. On a rate that can be changed, the node makes a copy of the dictionary and adds it to a list of all the recorded readings. Since the neural net needs more than one instant of data to be able to make predictions, this list of dictionaries is a good representation within python of time series data. The list also has a variable maximum size before it starts to remove the oldest entries and replace them with newer readings. Testing the neural net showed that anything less than around 90 instances of data wouldn’t allow it to make a prediction, so for most of the testing the 200 most recent readings at 0.25 second intervals were used.

In terms of design this functionality was decided on being its own node due to the amount of communication with mavros that was necessary, and the specific purpose it had. While this node was responsible for both communication and logic unlike the waypoint writing node, the functionality within this node is very specific to the information being read. This led to an overall very cohesive piece that didn't need to be abstracted or broken apart further.

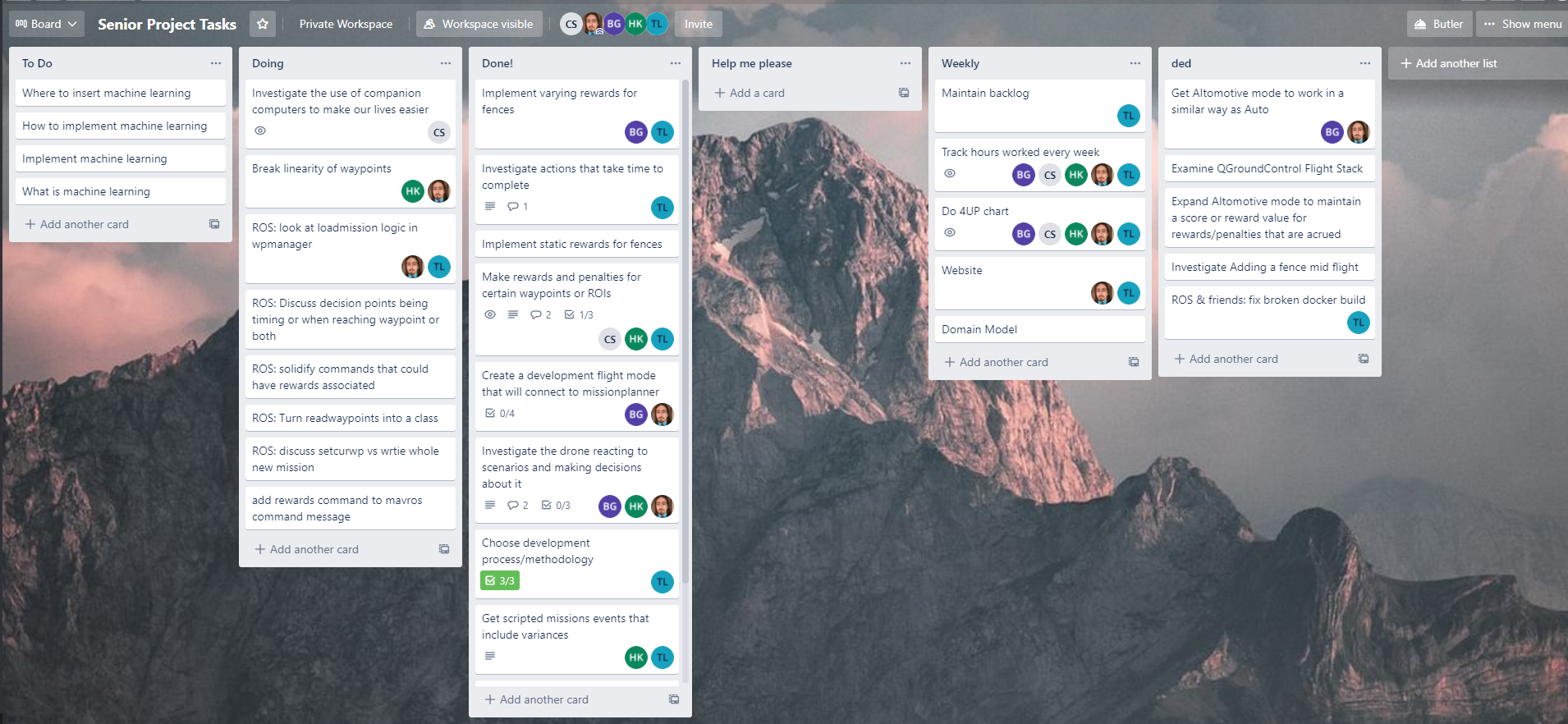
The final node is the driver node. It is responsible for reading all of the mission commands or waypoints from the drone and keeping an internal representation of them. When starting a mission the node creates a new waypoint manager class that takes the mavros mission data, and pulls out specific important pieces to build custom waypoint objects. These can be managed, tracked, and grouped within our system, which solves the exact problem that we initially had with ardupilot, and the reason we switched to ros. This node reads the mission every time a “decision point” is reached. For this project it was decided that decision points were every time the drone reaches a waypoint, but in the future it would be possible to have a decision point be based on time or a number of other things. Since this node has access to the manager with all of the waypoint data, this node is also responsible for passing that data to where a decision is made. There is a strategy pattern implemented so any python class with a “getNext” function can be used to decide where the drone should go to after reaching a waypoint. Then once a decision is made this node only has to communicate with the first mentioned ROS node about what was decided, and communication back to the simulation is handled by that node.

The strategy pattern that is used to make decisions isn't a perfect strategy pattern for a notable reason. Python is not a typed language so it is up to the user to use a correct type for everything. The system is architected so that it works well with an injected dependency of a strategy class that follows the paradigm, but in theory someone could break this as python has no strict requirement of what classes are used.

There exists in the codebase a few different strategies for making decisions for the drone. These currently include a random strategy that navigates the drone randomly, a smart strategy that navigates the drone to the waypoints with the highest rewards, not taking cost into account, and a utility strategy. The utility strategy is currently in an unworking state due to it not calculating cost correctly, but there are still a lot of working components to the strategy. The utility strategy is able to query the ernn and get the predictions to use in cost calculation. It attempts to calculate the distance to other points to use cost, but currently the distance calculation isn't correct. However there are safeguards that would work if the calculation was correct, such as having an infinite cost if the drone to go somewhere too far that would leave it unable to return to home. One final thing the utility strategy is accounting for is lookahead. When calculating cost and reward its important to look not just at the current location, but where the drone could go after that. For example the drone could go to one really far waypoint worth 5 points, or it could go to a group of three close waypoints worth two each. The utility function takes this into account by being able to look at waypoint costs recursively, and decides where the drone should go currently by looking at what could be the best path in the future. Lookahead calculations start to get very costly so the sponsor recommended not to look anything farther than three points into the future. Changing the amount of points looked ahead is changeable by a single variable, but currently the lookahead depth being used is only two.

### Process and Product Metrics

The team used a Kanban board to prioritize deliverables. Having a constantly evolving board allowed for us to maintain the level of flexibility that comes with agile through a constantly evolving backlog. We had a weekly 4-up chart which we kept up to date with for risk management and tracked the team’s hours spent on the project.

  
*The current state of our kanban board.*

Our deliverable plans were modified when we failed to meet deadlines, which should allow us to assess the accuracy of our estimates. We seemed to fall a week behind mid-way through the first semester, which kept us shy of our deliverables. In the second semester we vastly underestimated how long it would take to create a utility function and acquire training data.

We had vastly underestimated the proportion of research involved with this project. This resulted in low process adherence, because it lacked fitness for the work we were doing. At the start of the project, we figured that, since this was a low-code project for at least the first semester, the standard SDLC wouldn't be particularly useful. In retrospect, adherence to process could have been the difference between delivering what we intended, and what we have now.

On the topic of metrics, we did not maintain any. This was a poor decision in retrospect, but it was done because of the low-code expectation we had for development. The way we thought of it, there wasn't a lot; metrics could tell us, since the bulk of the actual work was acquainting ourselves with the technologies and domain. If we were to do this again, there are metrics we would focus on, however.

Process adherence metrics would have been very useful, especially given the WFH/covid situation. If we had kept track of how often daily standups were completed, or done planning poker, and maintained a minimum project velocity per timestep, we could have recognized when things like establishing a dataset to train on was taking too long, cuing us to reach out the our sponsor for help, or at least start investigating alternative options.

### Product State at Time of Delivery

At the time of delivery, the state of the project is significantly different from the product we intended at the outset. Instead of having a system capable of making on the fly path planning, we've built the tool that can be used to accomplish this. This is to say, our system has all the components to *make* path planning decisions autonomously, but it doesn't actually do this.

Due to the issues in establishing and utilizing a suitable dataset for the development of this project, we were never able to adequately train the eRNN to make actually useful predictions. Because the system cannot make those predictions, it also means we've failed to create a suitable utility function to be used in the decision making process. This is not to say what we've constructed is useless though.

If provided with an adequate dataset and utility function, the project should work as intended. All that's actually required to make decisions are those two things, and providing them would be relatively simple due to the architure we choose for implementing decision making. The way the drone decides to visit different waypoints is handled by an implementation of the Strategy pattern, so making modifications to the existing code base is as simple as it possibly could be.

### Project Reflection

As a team, we’ve had a number of successes and failures throughout the duration of the project. We feel that while we can revel in our successes, it is more important to learn from our mistakes. To comment on our victories, we had open and clear lines of communication with the sponsor and coach alike. This kept everyone on the same page as to where the state of the project was, in spite of the fact that communication was undoubtedly hampered by the COVID-19 pandemic. Additionally, we feel that we elicited requirements very well from the sponsor. Everyone on the team knew exactly what the end goal was, and this clarity was only beneficial to the team. We’d also like to address that the team managed expectations of the sponsor reasonably well, and would let them know when a feature wouldn’t be delivered on time due to poor estimation early in the process. This only adds to the good communication we had with the sponsor.

On the other hand, there were plenty of areas for improvement. First and foremost, we followed our process very tightly initially, and this improved the team’s performance, but this slowly but surely fell by the wayside. Daily standups would be missed, and the Kanban board would fail to be updated. This didn’t have major impacts on the team, as our productivity took only a small hit. The importance of following the right processes for the team is definitely more apparent to all of us.

One of the main areas of strife was estimation and time management, that we feel go hand in hand in our hindrances. The project is innately research based, and estimating the amount of time and effort required to gain an adequate understanding of a given tool or library proved incredibly difficult. We often undershot our estimations, and did not allot enough time for these tasks. This led to the team overall having less domain knowledge of the given system and its capabilities, and ultimately led to a complete system refactor to introduce a brand new tool at the end of the first term. Though added study of our tools and the tools available to us would certainly not be a silver bullet. For example we wouldn’t have known that Mission Planner randomizes voltage, instead of simulating a real battery until we ran into a problem where that suddenly mattered, which we eventually did.

This also made parallelizing work difficult. At certain points, some team members would be looking into what problems a given software would solve, while others were waiting for this work to be completed in order to be unblocked on their specific commitments. This is largely due to poor planning, and poor understanding of not what the system needed to accomplish, but how we were going to accomplish this. Instead, we should have put all available resources into looking into these tools and, and combining our research, and making an informed decision as to if we were going to use it, and how we were going to use it. This would have minimized research time, and maximized development time.