CS 4758: Bird of Prey
Autonomous Air Robot Vehicle Tracking

Brian Wojcik

Abstract— **Currently, the aerial robot is controlled by a physical controller the operator interacts with. Camera input into the aerial bot is used for object classification and avoidance. The focus of this project is instead to have the quadrotor autonomously track and follow a remote-control car, resulting in effectively no operator interaction. Based on visual input from its downward-facing camera, the aerial robot attempts to hold the car in its field of view and at a manageable distance by altering its horizontal and vertical translation & rotation. Both the aerial robot and the remote controlled car operate in physical areas lacking obstacles, such as a field or empty parking lot, so as to minimize and effectively eliminate any obstacle avoidance problems. This project examines the creation of an efficient end-to-end system to approach this problem (vision, analysis, and control), in addition to two types on controllers - reactionary and reinforcement learning.**

I. Introduction

Tracking and pursuing a vehicle is a real-world task assigned to Unmanned Aerial Vehicles. The potential of setting a robot after a target and have it follow, catch up to, or simply observe an object is very useful for a number of reasons, simplest of all being generally easier control from an operator's standpoint. Even if the aerial bot simply follows a land vehicle also controlled by the operator it allows the operator to work and focus on two dimensions instead of three.

There are two different approaches examined in this project. First, a simple 'reactionary' controller that, given a single frame of video input, reacts proportionally to the relative location of the car in the image frame. Rotation is based on the horizontal location of the car in frame, whereas speed of the aerial bot is determined by an estimation of nearness. The estimation of distance between the car and the aerial bot is done by number of pixels in the image tagged as 'car' (high pixel count -> car is near). The second controller is based on a policy learned by reinforcement learning. The scene is parsed into a binary string representing the state; it is then looked up in the policy table to find which action should be taken. Based on this policy the aerial bot can react in a number of ways, including altering speed, rotation, and horizontal translation. In both approaches, vertical height of the car is initialized on startup, and the aerial bot (currently) continues to operate on the height plane.

With either approach, the bot is able to consistently maintain a visual of the land vehicle. The is due, largely, to the aerials bot's ability to rotate quickly without necessary forward motion (unlike the land vehicle, which has a defined turning radius). Given this ability, the land vehicle is not able to out-maneuver the aerial bot, and should it succeed it making it outside the aerial bots field of vision, it is not sustainable.

One problem with the 'reactionary' approach is that there is no real memory of the land vehicles location. Should the land vehicle travel behind an object, the aerial robot with remain still, hovering and scanning the scene until the land vehicle re-appears. [For this project, we have defined the operating environment to be one devoid of obstacles, allowing the aerial bot can fly freely. The ability for the land vehicle to 'hide', however, is an interesting topic that is also examined.] By encoding previous actions / locations into the state-space of the reinforcement learning algorithm, the aerial robot can learn to precisely deal with this problem.

The second main problem faced is complexity and design. The reactionary controller's actions are based on hard-coded functions. Improving the reactionary controller requires the designer to craft complex, conditional functions. Although the bot will do only what it is told, this approach results in a large amount of complicated code rather quickly, where as the reinforcement controller does not. After an initial state-space and action-space design, the Air Robot's policy can be quite complex and function in ways not obvious to a human designer.

A. Related Work:

No projects were found that closely resembled this, however some do exist along the same vein. An aerial robot following a ground robot (that was not necessarily attempting to escape it) can be viewed here: http://youtu.be/WKnNSUbI-mk . A faster moving land robot (with the intent to escape) poses challenges not present in the above, linked, project.

II. Experimental Setup

Currently, all work has been done in simulation (due to a non-functioning robot). A large amount of a time has therefore been devoted to setting up the simulation environment in a practical, applicable way, that is conducive to how the aerial bot physically acts. This includes creating and scripting (in a 'visual' script language) a custom map in addition to changing the architecture of how existing aerial bot programs communicate (vision, analysis, and control). Setting up the simulator itself was very work intensive, requiring many files, programs, and packages to be downloaded and compiled to run.

A. Environment

The simulated environment (map) was created off a supplied 'test' map called 'AirRobotTestMap.ut3'. This map was edited and its launch settings altered so that it would launch with UPIS server that initializes memory for video feeds and control sequences to be shared between the simulation and the Air Robot control code. This map is a strong base to work in as it represents our operating environment quite well - an outdoor flat field on a clear, sunny day, devoid of obstacles (for the aerial bot). Subsequent versions exist with obstacles for the ground vehicle to hide behind.

B. Land Vehicle:

While running the simulator creates an AirRobot object in the simulated world, it does not have the ability to create a ground vehicle. By further altering the map described above, a realistic, fully functioning and controllable car was added to it. This is the object the aerial bot will track and follow. It functions similarly to a physical car, including how it turns and accelerates. One problem with adding the car to the simulation environment is that it does not stand out vividly from the environment, making perception difficult. The map was therefore scripted further to make the car stand out visually, changing its color upon spawn to a bright pink (a color chosen to stand out out well in the real environment/on the aerial bots camera). A filter was then coded so that any frame of video that aerial bot takes filters all non-land-vehicle colors.

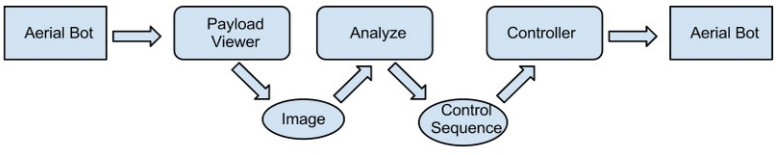


As an amendment to this project, perception was examined with a logistic classifier and a belief propagation (BP) network. Noted after the conclusion, the BP system results are quite similar to the current simulation – the car is overlaid with with a colorful estimation of its location while all other areas of the frame are dark. Working with this logistic classifier / BP setup, real-world perception can be handled efficiently and effectively, requiring only minor modifications to work in conjunction to this simulated perceptions setup.

C. Visual Input and Control Structures:

As previously described, the initialization work of the map and simulation environment has been completed. Both the car and the aerial robot are independently controllable in the environment, the car is scripted to stand out visually, and the AirRobot has access to its camera feed. The next step is to create the coding structures for image capturing, analyzing, and controlling based on this feed.

The original structures supplied with the simulator were 'PayloadViewer.c' and 'controller.c'. The Payload Viewer simply displayed and stored the most recent frame of video in memory. The Controller would move the aerial bot based on a constant, hard-coded loop of instructions. To implement perception based control, therefore, the most up-to-date frame was read from memory; it was analyzed by the analyzer; a control sequence based on this analysis was saved to memory; finally the control sequence was read and acted on by the controller. Visually:



This process is not optimal. Complex interactions occur between these programs which all function independently. Each program (PayLoad Viewer, Analysis, & Control) all have their own sleep timers, their own overhead, and have to be synced with the other two programs. Problems with communication arise where Analysis could re-read the most recent image if no new one has been saved, effectively making the aerial bot act twice when it should act once. Similarly, the controller could re-read an old control sequence multiple times if analysis is not fast, or missing one is Analysis is too fast. Simply put, the initial process architecture would not work.

The solution to this problem was to instead combine all the sections into a single file, a single master program. In the new setup, the overheads of saving/reading files & images do not exist, and only one sleep timer exists. Additionally, more information can be passed between the stages, such as Analysis passing a multi-sequence chain of actions to the controller. This 'master' program is more efficient and faster than the old architecture while possessing all the same abilities and more.

III. Approach

A. Reactionary Controller

1. Design

Given a frame of video, the controller code controls the aerial bot in a reactionary process. The process is based on the location of the land vehicle within the quadrotor's field of view. Two methods of movement are utilized by the aerial bot: Rotation, and Forward / Reverse translation. The operating height is initialized on startup and the aerial bot remains at that level throughout operation. Rotation is a function of the horizontal location of the land vehicle within the aerial bots field of view. The formula being:

Rotation Action = 2 * (x_loc% -0.5)

Forward/reverse translation speed is a function of the car's distance from the aerial bot. The distance metric used is based on number of car pixels observed (directly related to the distance of the bot). The formula used is:

Speed = (Max_Close-pixcount) / Max_Close

where Max_Close is the distance in pixels the Air Robot should stop moving closer. A pixel count beyond Max_Count results in a negative speed, moving the Air Robot backwards.

2. Advantages

The advantages of this controller is that it is clear what the aerial bot is doing in all states. This design is easily readable and adaptable (to a point). Possibly the most advantageous aspect of this approach is that the action space is not discretized - actions can take on rotation or speed values anywhere from [-1, 1]. Rotation and Speed actions can also easily be calculated independently and then superimposed to make complex actions.

3. Limitations

This approach is limited, however, by the designer. The functions are chosen by trial and error to find valid action performance, and are based heavily on the developer's thought process and expectations. After a base level of simple action functions, improvements to this controller become difficult. Rarely used yet complex conditional functions must be included to produce improvements in edge-cases. Initial ease of implementation is followed by a series of complex conditional actions for specific state occurrences. The code is no longer easily understandable or adaptable.

B. Reinforcement Learning Controller

1. Design

The general design of the reinforcement learning controller follows:

- Design a state and action space that the Air Robot will operate in.
- Randomly act these actions in the simulator while the ground vehicle moves.
- Record state transitions and average rewards based on these observation points.
- Complete value-iteration reinforcement learning to generate a policy for the Air Robot.
- In live execution, the Air Robot classifies its current state and looks up the action for it from the learned policy.

2. Simple State and Action Space

A simplified state and action space was created to take advantage of a low amount of required training data. This included a state space of : 3 depth positions and 3 frame positions

represented as 4 bit binary string. The action space included 3 speeds and 5 rotations. Due to this limited space, the training data required to fully examine the space is 2048 training points (calculated by 8 actions * 2^4 bits for start state * 2^4 bits for result state), an amount that is perfectly reasonable to obtain.

The benefit of this limited controller is that it is a strong base for future learning. It is easy to gather an excess of training data, and is easily extendable to include a more complex state space or action set. For the most part, this set up is directly comparable to the reactionary controller, in that it utilizes a similar action space (though not discritized as it is here). By comparing this controller with the reactionary one, and their performance, one design can be chosen depending on desired results.

3. Complex State and Action Space

In addition to the simplified reinforcement learning controller, a theoretical reinforcement learning controller was also designed. This design differs in that:

- State space now include 3 distance states, 3 horizontal percentage of frame states, 3 'past' states (previous horizontal percentage), and 3 relative velocity states.
- State space is represented as an 8 bit binary string.
- Action space now includes 3 speed options, 5 rotation options, and 2 horizontal translation options.

The result of this design is that now the state space fully encodes both the aerial bot's state and the ground vehicle's state. By looking at the aerial bots past rotation, as well as the ground vehicle's current horizontal position in the frame, the state space encodes the car's translation perpendicular to the air robot's line of sight. This information tells the bot if the car is moving to the left or the right, and how fast. By keeping track of the relative velocity between the car and the air robot, the state space encodes the car's translation parallel to the air robot's line of site. This information tells the bot if the car is coming towards or away from the bot. As this state space encodes how the ground vehicle is moving in relation to the Air Robot, this set up is extremely powerful and will produce a precise, smarter policy. Additionally, this complex setup includes more actions to fully utilize the range of actions available to the physical aerial bot.

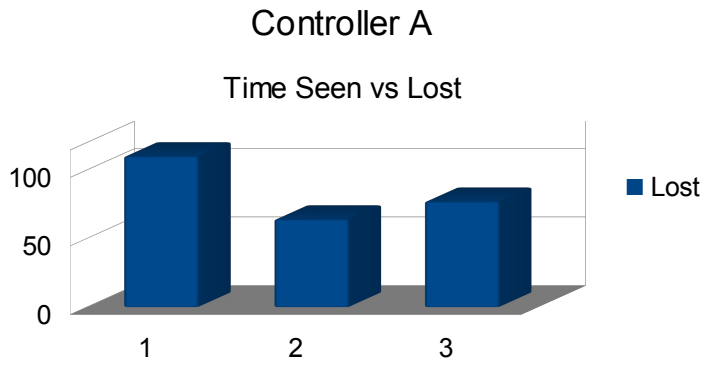
This setup was not used, however, simply because of the massive amounts of training data required. For 10 actions * 3^4 initial states * 3^4 resulting states, a total of 65,610 data points are required to observe the entire space a single time. For this application the required training data is far too great, however this setup provides a theoretical near-optimal design.

IV. Results

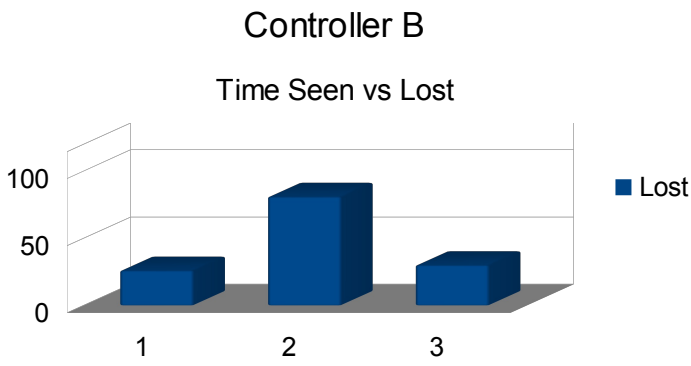
Both the reactionary controller and the 'simple' reinforcement learning controller were implemented for this project. Both achieved comparable qualitative results by observation, following the ground vehicle in simulation for an extended period of time. The reactionary controller ultimately outperforms the RL controller at this stage in development as it's action space is not descritized (a key benefit of the reactionary controller).

Quantitative results are in agreement with this finding. The controllers were evaluated based on the number of frames in which the ground vehicle was 'lost', or not in frame, for a given run of a timed simulation 500 frames in length. This evaluates both the controller's ability to keep the car in frame, as well as its ability to re-find the car when lost. The ground vehicle was human controlled, and driven between sequential way-points to limit driver-bias.

The reinforcement learning controller lost the ground vehicle an average of 84 frames out of 500 frames. The data for three runs are presented below.



The reactionary controller lost the ground vehicle an average of 46 frames out of 500 frames. The data for three runs are presented below.



V. Conclusion

The result of this project is the creation of a complete, end to end, system for vision-based action of the aerial bot. Additionally, multiple controllers have been examined, implemented, and tested in various forms.

The creation of multiple controllers has had interesting results. The reactionary controller out performs current reinforcement learning controllers in tests for numerous reasons. The first is action space discritization. The reinforcement learning controller relies on a finite number of actions while the reactionary controller is unlimited. Even alternative learning algorithms like Value Function Approximation can not solve this problem - the action space must still be discritized.

While action space discritization is a problem for the reinforcement learning controller, ultimately the reinforcement learning controller is expected to outperform the reactionary controller. The reinforcement learning controller is operating on a severely limited action space, state space, and number of training data points. As any one of these increases, large improvements in the operation of the RL controller will be seen. The implementation of the reinforcement learning controller is in its infancy - there is room for improvements in nearly all aspects of its operation, where as the reactionary controller is at its functional limit. Additional effort put into the reactionary controller will have limited improvement on its operation, and will increase code complexity noticeably. Through the examination of both types of controllers - their development, improvement, and results - it is concluded that, for any implementation beyond the basic, a reinforcement learning controller should be implemented between the two.

VI. Amendment: Perception

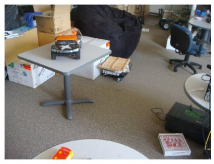
As previously noted, the perception system of this setup was expanded to use a logistic classifier and belief propagation (BP) network. Coded in simulation, the ground vehicle was coated in a vivid overlay to make perception easier. In the physical world, however, this technique is non-functional. It would require the ground vehicle to be similarly coated in a visually vivid, unique color and remain under constant lighting conditions. Should the color not be unique, color 'noise' from other objects would influence perception. The ground vehicle traveling under different light intensities/sources would cause its perceived color to change, again making this approach to perception difficult in the physical world. In an effort to deal with these problems, perception was redesigned to not be purely color based. The perception features were instead learned by a logistic classifier and resulting weight set fed through a BP network, resulting in a robust, accurate visual perception system.

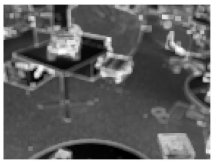
A. Complex Environment

The logistic and BP classifier was initially trained on data taken from a 'complex' environment. This environment was an indoors robotics laboratory. Due to the similarity between the ground vehicle and spare parts from other robots and experiments, should the classifier strongly identify the vehicle in this environment, it can be extended to operate in its simpler, expected, operating environment of the ground vehicle outdoors. Training in the complex environment raises the confidence that the perception would correctly classify the car, and only the car, in outdoors environment where other metallic objects, or even objects with wheels, might appear unexpectedly and not be represented in the outdoors train/test set.

1. Logistic Classifier

Trained on 'complex' data, the logistic classifier achieved accuracy rates of 83.0% . Displayed below, the first image from the left is the original scene. The second image displays the sign of each area's classification, white for positive, black for negative. The third image is a superposition of the two; areas in the image more strongly black or white show the confidence of that classification. As can be seen, the car is largely colored bright white, indicating a strong confidence in its (correct) classification. Other areas of the image, however, are also colored while. This noise / misclassification is one of the items the BP network aims to correct.

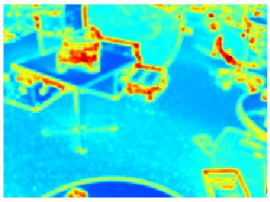
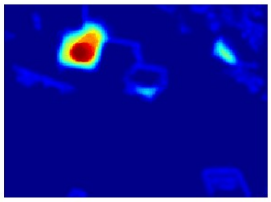




2. BP Network and Classification

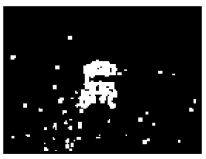
The goal of the BP classifier is to clean the results of the logistic classifier. Functioning optimally, the output from the BP classifier removes erroneous positive classifications while extending the correct classification of the ground vehicle to the ground vehicle's correct shape. The BP network is a series of interconnected nodes, similar to artificial neural net. By pumping each node the output of the logistic classifier, each node builds up its own potential until in meets a defined threshold. Once that threshold is reached, the node fires, increasing the potential of its neighboring nodes. The image pixels, then, are classified by the rate/number of times they fire; a higher firing is more confidently positively classified. By altering how these nodes interact, when they fire, and other parameters, the BP network can effectively remove all noise from the input image, and extend the classification of the car to solely the bounds of the car.

Running the BP classifier in the complex environment, a recall rate of 94% and a precision rate of 51% were achieved. As can be seen in the below images, the resulting classification (right) is a cleaned version of the initial (left). Less erroneous points are misclassified, and the car's classification 'bubble' more accurately represents the car's true shape.

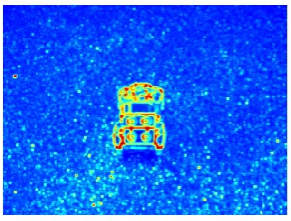
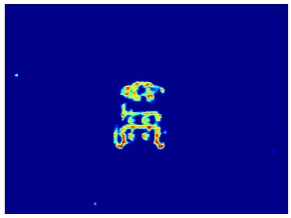



B. Simple / Expected Operating Environment

As expected, the logistic and BP classifiers' parameters could be precisely extended to the more simple, expected operating environment. The ground vehicle is expected to operate outdoors, either on grass or on open blacktop pavement. Using the same parameters (downsampling rate, feature patch size) the logistic classifier achieved an accuracy rate of 88.6%, well above that of the complex environment. As can be seen below, however, noise still exists in the data even at this higher accuracy, and can be filtered out by the BP system.


The results of logistic classifier further increased the precision of the BP classifier results, achieving (on average) a 99% precision rate and 11% recall rate. Although recall is a low in this setting, the previously described controllers are dependent on an accurate location of the car, which this does quite well. It dramatically reduces noise in the data while keeping relevant car data intact (or even enhancing it). This functionality uses the exact same node parameters as the 'complex' lab setting, showing that this BP design correctly identifies the car (and only the car) in both the lab and outdoors setting; a quite powerful ability.

C. Final

The use of a logistic classifier in conjunction with a belief propagation network creates an accurate and robust perception system for identifying the ground vehicle. Not purely based on color, it provides distinct advantages over the base perception used in simulation, and promises greater ability to identify the car and only the car. The fact that the same BP network can be used for both indoor and outdoor environments without

alteration means that it has rather few weak points, functioning well in a wide range of environments and operating conditions. By removing noise while preserving or enhancing the correct data, it is a very accurate and useful system. Paired with the aforementioned control code, a complete end-to-end system for finding and following a ground vehicle with the aerial bot has been created and examined.