Computer Science 189: Autonomous Robot Systems
Final Report

Julia Pearl and Juliet Nwagwu Ume-Ezeoke

May 10, 2019

Whether it be companies like Amazon who use robotics to make warehouse operations more efficient, or Fetch Robotics, which focuses on using robotics to improve the profit margins of other companies, it is clear that many enterprises are taking advantage of robots' unique ability to do work autonomously. The implementation of this final project for CS189 offered a glimpse of what the process of developing such systems are like. The goal of the project was to write a program that would allow a robot to fetch candy when told which dispenser to get candy from. The project was implemented in three parts. The first was parking, ensuring the robot could successfully place itself under a candy dispenser with enough accuracy to catch all the candy inside a bowl placed on top of it. The second was moving between different dispensers, using a navigation system based on knowledge of the ARTag locations. The final piece of the project was to avoid obstacles successfully, making sure that the robot could still get to its desired location regardless of whether or not there was an object blocking its originally planned path.

The overall approach to the project was to use the robot's knowledge of its home EKF-position and the EKF-position of the ARTag to decide its path to a given dispenser. The robot would move toward the EKF-position of the ARTag until it recognized the ARTag and was closed enough to it in order successfully park. This would then activate the parking sequence for the robot, allowing it to geometrically plan the next steps it would have to take to place itself under the candy dispenser and wait for candy to be dispensed. After about ten seconds, the robot would then return to its home position and wait for another command to tell it where to go. To handle the occurrence of obstacles in the robot's planned path to the EKF-location of the ARTag the robot used the depth sensor. The depth sensor was able to identify objects that were not ARTag within a certain distance of the robot and send a message to the main program to tell the robot that there was an obstacle to be "avoided". Whether or not the robot would avoid the obstacle was dependent on how far the robot away was from the dispenser. If the robot was quite far from the dispenser, it would pause momentarily before slighting adjusting its direction to veer out of the path that was being blocked by an obstacle. If it was closer to the ARTag, it would pause for a longer amount of time, acting under the assumption that there was another robot in the location of the ARTag.

The limitations of the strategy were that the robot would sometimes have difficulty finding the ARTag, in which case it would go completely off track. This often meant that the camera recognition in the roslaunch file was not working correctly and the ARTag would not be seen. To

neatly handle the unpredictability of the roslaunch, a sequence could have been implemented that would allow the robot to return to its home or pause and spin when the EKF location had been reached, but the ARTag had not been seen. Additionally, the parking sequence of the robot was one that was meant to handle the robot approaching ARTag from all angles, and ideally all distances. However, since the ARTag could only be seen at a certain distance away (distances that were eventually hard coded into the program) it would have been a good idea to spend less time on this functionality, and more time fine-tuning the navigation between the dispenser themselves.

Other than these issues that were mostly based on more neatly handling quirks of the hardware, the strategy that was implemented was quite reliable. It met the specifications of parking, travelling between different dispensers regardless of what its home dispenser was, and handling other robots or obstacles in its way.

Milestone 1

The strategy for implementing Milestone 1 was heavily based on the specification that the robot should be able to park from all angles and ideally all distances. Realistically, the robot could only reliably identify the ARTag from within approximately 1.5 meters. This meant that the code deciding which dispenser the robot was going to had to be altered such that the robot would only begin parking when it was within a certain distance from the ARTag.

The overall strategy for parking the robot involved implementing a finite state machine discrete from that of the main run function that would guide the robot to successful parking. The ready state in the code is active when the robot has just transitioned into the parking sequence. The main sequence ensures that the robot only calls the parking sequence when the robot is within a certain distance of the ARTag, so the robot should immediately enter the ready state upon the initialization of the park function. In this state the robot simply renames for variables to make the code easier to understand. One important note is that for most calculations, rather than directly using the variable returned by the Alva Markers orientation parameter, the complementary angle, called beta, is used.

When the robot has successfully located the ARTag, it zeroes the x-distance between the ARTag and itself, such that the robot's camera will be directly pointing at the ARTag. It then computes a scalene triangle that is described by beta, the z-distance between the robot and the ARTag, and a goal distance, called LL_DIST. LL_DIST is the distance away from the ARTag that the robot should be before it can orient itself to face the ARTag and move to successfully park.

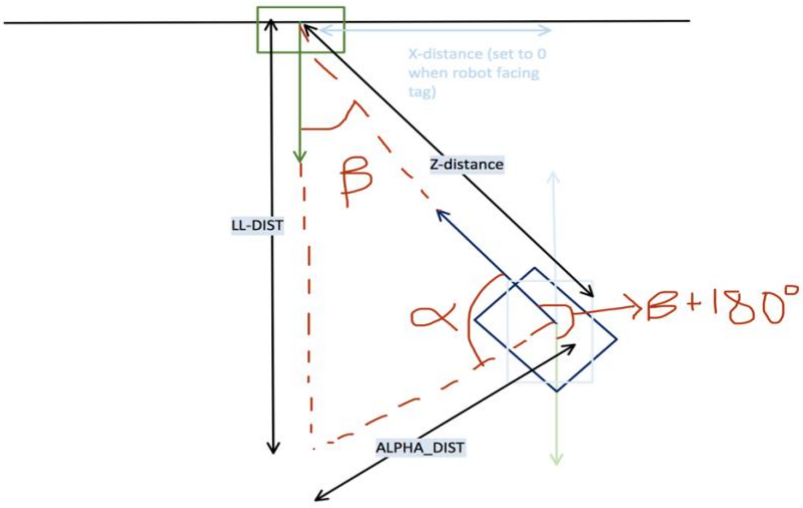

Figure 1: Diagram of Important Parameters in Parking Strategy

From knowing this information, the program returns and angle, alpha, and another distance, ALPHA_DIST. The angle alpha is the angle that the robot needs to turn in order to be able to travel across the top of the scalene triangle. This angle is calculated using a modified law of cosines function that takes in three distance and outputs the angle between the first two inputs. The distance ALPHA_DIST is how far the robot needs to travel after turning alpha. This is calculated using another modified law of cosines function that takes in two sides, LL_DIST and the z-distance, beta, and then returns ALPHA_DIST, the length of the top of the triangle. Both of these parameters are calculated only when the x-distance has been zeroed, as they correspond to the triangle at the specific z-distance and beta that the robot has at that instance.

Once the robot has turned alpha, and moved ALPHA_DIST, it should be theoretically be at a location where turning to face the dispenser and moving towards it will result in the robot being placed under the dispenser within an acceptable tolerance. In reality, the alpha that is computed might be slightly different than the alpha that is actually turned by the robot. The same goes for ALPHA_DIST. This means that by the time the robot is oriented towards the robot it go underneath the ARTag at too great of an angle.

Initially this issue was addressed by having the robot run through the sequence again if it's the absolute value of its beta was beyond a certain range surrounding zero degrees (the magnitude of beta is zero when the robot is right in front of the ARTag). This meant zeroing the x-distance again, recalculating and executing alpha and ALPHA_DIST, which would increasingly become smaller, until the beta was within a good range. At this point the robot would be in a position right in front of the robot such that it would be able to reorient itself and reliably park.

It was discovered that, due to the variability of the beta values from the ARTag, there was too great a probability that the robot would calculate the incorrect values of alpha and ALPHA_DIST. It would often be the case that the robot would be right in front of the ARTag and then turn by an extremely wide angle and move a large distance away such that the ARTag would no longer even be in view.

This behavior was addressed in two ways. Primarily, when the ARTag had traveled ALPHA_DIST, before zeroing the x-distance, it would check that it had valid data from the ARTag. If the last 20 x-distances that had been reported by the Alva Markers module were the same, tit meant hen the robot was no longer seeing the ARTag, and the program would enter a lost state. In this mode it would oscillate slowly while seeing if it could relocate the ARTag. If it had been in the lost state for too long, the park function would return unsuccessfully. The main function called park at a time when it was already moving towards the EKF position of the robot, so the park function would be called again with the robot in a more optimal position.

The next way the behavior of moving away from the ARTag was addressed was by limiting the valid alpha angles that could be returned. This can be seen in the cool_math.py script. In the function that returns alpha, if the value of LL_DIST was greater the the sum of the squared z-distance and ALPHA_DIST, then a null value was returned. This prevented the robot from turning too far away from the robot. In retrospect this might have contributed to inaccuracies in the system since the robot was then no longer calculating valid alphas for some of the possible locations of the robot. If there was more time to work on the project, a better strategy would have been to always return accurate alpha values, turn alpha, move an ALPHA_DIST, and then have the robot turn by the most recent beta value to generally face the tag before zeroing its x.

Eventually, after dealing with the all too frequent occurrence of the robot going completely away from the ARTag, it was decided to use a boolean to reflect that the robot was almost ready to "move perfectly" into the dispenser when the robot had successfully moved ALPHA_DIST. This meant that the robot would only go through the sequence of trying to get an optimal beta value only once and then move towards the ARTag at a slightly off beta value. However, this was resolved by having the robot zero its x-direction again once it was closer to the ARTag. When the robot gets to be underneath the dispenser, it then initiates a sequence of sleeping for 10 seconds, backing out, and returning to the main function that the park was successful.

Milestone 2

Our overall strategy for milestone 2 was to navigate using EKF positions until the ARTag desired was in vision.

Mapping EKF positions: We distinguished the robot's starting EKF position to be (2, 15) based upon the setup of the testing area. When running the code for milestone 2, the desired ARTag was passed in as a parameter. The id for each ARTag had a corresponding EKF position in a python dictionary at the top of the file. Those EKF positions were calculated using measuring calculations of the testing area. With the EKF position of the desired ARTag and the current EKF position, the robot would calculate and turn the angle necessary to travel straight to the desired ARTag. After turning that amount, the robot would move forwards.

Identifying correct ARTags: When the desired ARTag was passed in as a parameter, the function that processed ARTag location only would search for the desired ARTag. While the robot was

travelling in the direction of the ARTag's EKF position, at some point the robot would see the desired ARTag. The robot would then stop moving forwards and proceed to call the park function from milestone-1.

Returning home: After the robot was successfully parked, the robot would change its desired ARTag global to be ARTag 1 or the "home" ARTag. The code would pull the EKF position of the home tag from the dictionary. The robot would do the same turn-angle calculation and forward movement in the direction of the home ARTag, and then it would call the park function when the ARTag was close enough to be processed. The robot would then park at the home ARTag and wait for its next function call.

Cases 5-6, when obstacle was in the way of the ARTag: For ARTags 5 and 6, an obstacle (the table) was in the middle of the route between the home ARTag to the 5 and 6 tags. To avoid the obstacle, there were three more ids added to the dictionary of the ARTag's EKF positions: 11, 51, and 61. When the robot has the 5^{th} or 6^{th} ARTags passed in as a parameter, the robot multiplies those ARTags by 10 and adds 1 to calculate the new id that corresponds to a position around the obstacle that has no actual ARTag. The purpose of the new position and the new id number is so that the robot will avoid the ARTag and be able to calculate, from the new/fake id, the ar_id that it initially was sent to. After the robot heads in the direction of the new/fake ARTag for a certain amount of time, the robot changes its desired ARTag to be the initial desired ARTag (5 or 6), and, because the robot has successfully cleared the obstacle at that point, is able to find and park at the ARTag. Then, instead of returning home by changing the ARTag to be Home or 1, it changes the ARTag to be 11 (1 * 10 + 1), another fake id, so that it can clear the obstacle and then return to the home ARTag.

Testing: Our implementation of parking was not correct until 3^{rd} week. We did not know how correctly the return action of the robot would function, especially given that EKF positions become less reliable the longer the program is run. Our main concern was that we would program the robot to return back to a certain EKF position, and the robot's runtime would have altered its own EKF position too much for the robot to be able to recognize the home ARTag on its return path. However, we figured that the robot would have a large threshold for error until it would not be able to recognize the ARTag.

Milestone 3

We decided to tackle collision avoidance by defining our path of logic.

1. The regular clear path would not trigger the avoid_obstacle state:
When there are no other robots on our robot's path, the path should be clear.

2. If, during the travel on the EKF path, the avoid_obstacle state were to be triggered, then another robot must be in our robot's path:

The travel on the EKF path means that we are not near our destination of an ARTag. Another robot is not trying to be park but is just in our way. Because the robot's path is calculated by EKFs, once our robot moves out of the other robot's way, our robot will continue to go to the ARTag. So the robot was programmed to turn until no obstacle was in its vision in the direction away from the obstacle and then it was programmed to move forwards in that direction before returning to its "go to ekf" mode. The robot's vision was narrowed so that it only cared about objects directly in front of it.

3. If we are moderately close to an ARTag, and we see an obstacle, it's another robot parking. Moderately close to an ARTag means that we have called the park function. We will wait for a long amount of time (15 seconds) because another robot is parking.

4. If we are extremely close to an ARTag, we are going to have to "claim our spot."
If we see another obstacle, that could be another robot or that could just be the wall.

5. Any bumps will result in another robot coming at the robot from an angle not directly in front. It would be best to sleep for 5 seconds or 15 seconds, depending on how close the robot is to the ARTag, because moving backwards is not guaranteed to be obstacle free.

Final Demonstration Assessment

The performance of the robot during the final demonstration was not completely expected. There had been some changes in the code right before the presentation, which meant that the EKF positions that the robot was trying to go to were not completely accurate. This meant that the robot required a bit of nudging to find the dispensers, which was not the behavior that was expected after resolving these issues the night before. There was also the usual hardware issue of the robot not being able to see the ARTag, which is why when it initially ran it seemed to making a beeline for the wall rather than ARTag 4, which was the goal dispenser at the time.

However, once the roslaunch issues were sorted out, the robot was able to reliably go to the stated ARTag, park, and return to the home station. The first time it parked, at dispenser 4, it seemed to be a little further away from the dispenser than normal, which was unexpected behavior also.

The strategy for avoiding obstacles was not demoed on the robot due to running out of time, but there is video of avoiding obstacles included in another part of this submission.

To improve on the current strategy for fetching candy, a good idea would be to use waypoints on the map when navigating to dispensers. Rather than having the robot navigate all the way from its home base to the dispenser, it could navigate from the home base to a given waypoint along the way to the goal dispenser, and the next waypoint after that. This would allow the robot to have a better idea of where it is going at all times. It would also decrease the chances of the robot going off in random places when it cannot find the ARTag. Finally, this strategy would allow the parking function to be more accurate. If there was a waypoint on the map right in front of the ARTag, the robot would generally start off right in front of the ARTag. This would decrease the

correction that the robot has to make in order to reach an ideal beta value before parking, and generally make the parking far more accurate.

With regards to parking, some improvements to the strategy have been discussed on the section on milestone 1. However, another improvement could be made to decrease the angle that the robot went under the dispenser with during the final demonstration. This to decrease the range of values in the z-distance for which the robot was zeroing its x-distance as it approached the ARTag. Rather than having the robot zero the x-distance continuously when it is close to the robot, it should zero this distance only once or twice.

Discussion of Ethics

The human jobs left at the candy store would be the occupation of restocking the candy dispensers, keeping track of popular candies and constantly adjusting the inventory to match user demand, monitoring spilled candy, and greeting customers and making them feel welcome.

When considering the value of income, workers would be paid appropriately for their time. Seeing as the job of providing candy to customers would cost significantly less to the company since it has been automated away, there would be more room to pay all workers more generously. For some of the jobs, there would also be high equity in between workers and the realization of their value with regards to developing one's skills and building community for some of the jobs. For a person who is interested in using data analysis techniques to turn raw data generated by the robot's dispenser activities into income for the store, the job of accurately choosing the restock of the store would be both challenging and rewarding due to the fluctuation of users' palettes. The job of being a greeter has clear implications for developing community, as since more mundane tasks have been automated away, there can be a greater focus on making customers feel welcome. However, for tasks such as restocking dispensers or monitoring spills, there is less room for personal or community development.

Following from this assertion, it is easy to see why the most serious inequality would be found in the more mundane jobs of restocking the dispensers or cleaning up spilled candy. Although workers would be well paid, they would likely be less well paid than the data analyst who records trends at the candy store and makes purchasing orders based on trends. However, in terms of the functionality to the store's success, both jobs are of high importance. Additionally, while the data analyst has the opportunity to grow in this valuable skill set, those tasked with cleaning candy have no such benefits. Clearly, this set up is not acceptable under Rawls' principle. Not only are the social goods unequally distributed, they are also distributed in a way that further disadvantages those who are worst off.

One change that could be made to reduce this inequality would be to add work that contributes to personal growth in the job description of those employed for more mundane jobs. For example, rather than having separate jobs for the greeters and those keeping the candy store physically in order, they could be combined into one job. This would allow there to be far more variety, social interaction, and personal improvement in the work of those whose job it is to essentially clean up the store. It would also be of greater use to the store, as those who are restocking the candy have a very intimate understanding of how the store is organized and can

relay this to customers as the robots retrieve candy for them. This setup would be more palatable to Rawls as there is now a more equal distribution of social goods. Both jobs that require analysis and those that involve mundane tasks such as stocking dispensers are paired with the social goods of personal and community development. As those tasked with cleaning up spills and restocking dispensers would have the added value of being greeters, there would also be a more equal distribution of resources.