# Target Identification and Retrieval Integration with Swarm Robotics

Kumar Yogesh Shah, Naadir Kirlew, Daniel Klumpp, Jose Tormo, Grant Wilcox, Carlos Celemin, Abdulaziz Alenezy, Pietro Dimitri, Ibrahim Tansel

Florida International University
Department of Mechanical and Materials Engineering
Miami, Florida 33174

## ABSTRACT

The research presented within the following paper was developed in conjunction with the goal of participating in the 2018 NASA Swarmathon Competition, an In situ resource utilization type challenge. FIU's motivation for participation was the advancement of research with a purpose of developing integrated robotic platforms through means of swarm robotics. The FIU project team conducted research via a variety of scientific papers and research on swarm robotics and behaviors, as well as various search algorithms and their performance to ultimately develop a search algorithm code to complete the desired search and retrieval task within the confines of resource availability and time constraints. The search algorithm was implemented with identification and retrieval operations, then tested against a base code offering limited functionality. The two variations were tested via the NASA provided virtual platform, Gazebo. The results of the two simulations were used as comparison to gauge the margin of improvement achieved by the developed search and retrieve algorithm code over the provided base code. Results are outlined below.

## I. INTRODUCTION

The motivation for the research conducted and described below is the Third Annual NASA Swarmathon Competition held annually at the Kennedy Space Center in Florida. Florida International University's (FIU) objective was to develop an integrated robotic platform capable of improving the resource retrieval rate by upgrading the base code and search algorithm used in the swarm robot rovers provided by NASA. The reason for the development of these robotic platforms is resource location, identification, and collection, also known as ISRU (In-situ resource utilization) in extraterrestrial type missions. Using the NASA provided robotic rovers, dubbed Swarmies, the team aimed to develop an efficient search and retrieve algorithm to autonomously collect the available "planetary resources" which are represented by AprilCubes in the Swarmathon competition. FIU was motivated to participate in the Swarmathon to contribute to the collective cause of revolutionizing extraterrestrial exploration via implementation of swarm based robotic systems through research, innovation, and teamwork.

The study had two components: coding and enhancement of it with by reviewing previously published studies. The basic code was prepared by dissecting the base code provided by NASA, provide a basic understand "as-provided" functionality, as well as exploitation possibilities regarding the modification of the provided base code. At the same time existing publications related to swarm functions including algorithms, effective ways of communication between rovers, as well as optimization algorithms were reviewed. From the publications and research, a series of possible search and retrieval behaviors were designed and studied to understand its functionality on the Swarmies platform. The final code was prepared by integrating the parallel studies. Testing was then performed, and the results of virtual simulations were compared to analyze the outcomes.

## II. LITERATURE SURVEY

### *2.1 Swarm Robotics*

The term swarm robotics is based on multi-robot systems referring to multiple robots cooperating to perform a complex set of tasks. Swarm robotics is an approach to multi-robot systems in which multiple robots are involved, taking biological behaviors of swarm like creatures, such as ants, as inspiration to accomplish search, identification, and collection. Swarm robotics dates back nearly 40 years, to the 1980's [2], having had major strides as research provides new solutions for complex tasks in a broad field of possibilities, yet swarm-like technology is still considered to be in its infancy stages.

Due to the availability of the multiple agents, the swarm itself may be far more simplistic than a centralized single robot performing complex tasks, therefore, the behavior of the swarm would be considered a complex combination of many simple individual robots. Swarm robotics allows for multiple agents to take on complex tasks, allowing for a simplification of each unit rather than a single, far more intricate robot. The decentralization and autonomy into local communication through distributed intelligence between each swarm robot allows for a greater overall grasp of the complex task at hand, leading to an increased robustness and reliability regarding the task at hand [9].

Based on the task and its complexity, sheer numbers alone do not necessarily produce the best possible outcome according to insight gleaned from preliminary research. Swarm robots need an efficient way of communication with a certain level of intelligence to reliably cooperate as a whole. Multi-agent robot systems, such as swarms, are intrinsically advantageous over the single robot system

solution [1]. Advantages such as parallelism allow for simplification of a larger single task into subtasks for increased speed and or efficiency, as confirmed by related research and applications. The robustness of the solution is increased due to the introduction of redundancy by essentially eliminating single failure points. Adding to the advantages of swarm technology, adaptability and flexibility allows for tasks to not be exclusive to anyone agent in case of unexpected failures. Lastly, the scalability of swarm robots is desirable as tasks assigned to multi-robot systems become far more complex, thus the high capacity for scalability becomes increasingly relevant.

## *2.2 Robot Operating System (ROS)*

The Robotic Operating System (ROS) is a flexible platform to implement coding for robots that can be shared and communicated as needed across multiple platforms in the robotic swarm. The ROS includes libraries, tools, and convention that aims to assist in creating complex robot behavior [9]. The ROS is an integral piece of the swarm technology necessary to facilitate autonomy of the search and retrieve competition goal. As the Swarmies navigate the competition course, communication related to obstacles and resources can be passed between the individual rovers via the ROS. While the communication capabilities were not utilized within the subject experimentation, the ROS provides the possibility to expand the search and retrieval capabilities of the robotic swarm.

## *2.3 AprilTags*

The AprilTag system is a printable code style tag that is used in computer based visual detection systems. Edwin Olsen at the University of Michigan founded and pioneered the AprilTag system. The tags are 2D barcodes that were developed to aid in the visual capabilities of the robots as they perform robotic functions. See the figure below.

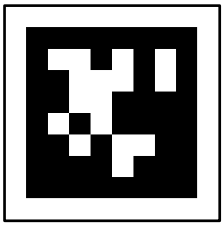


Figure 1 – AprilTag 2D barcode

The tags provide a unique fiduciary marker for the robot's camera that facilitates the tag's identification and determination of the distance to the location in the image. It is advisable that the user calibrates the camera and pre-determines the physical size of the tag in use. The tags are made in a normal printer while the detection software is used to calculate the approximate location based on the known size of the unique code arrangement. Java or C applications are used with AprilTags; however, the C implementation is recommended since it requires no external application [4].

## *2.4 Search Algorithms [8]*

The following represents search algorithm techniques researched by the FIU team as code design alternatives for implementation into the ROS on-board the robotic rovers. Each algorithm was studied to assess feasibility regarding the ease of implementation considering time constraints faced by the team in order to develop a working algorithm necessary to compete. The algorithms that offer advantages over alternatives are favorable but require further resource input to achieve successful coding. FIU identified the following algorithms as possible candidates for implementation.

Path finding algorithms display maps as nodes with each node having a movement price associated with the potential movement. This price represents the value tied to the cost of moving to a specific tile while calculating the path. While calculating a path, these algorithms take into account the local and global price to make sure the path chosen is the most optimal. Local price is the price of associated with movement to each of the tiles available around the current position or source tile. Global price is the total price associated with traversing each path to the intended goal. Analyzing both permits the code find the most optimal path to the targeted goal.

### *2.4.1 Greedy Search Algorithm*

The Greedy code takes into account the local costs while moving to the goal hoping it would lead to the cheapest global cost. This as shown in the image below does not take into account for objects in the way and makes the path route sometimes more expensive than other route plotting algorithms.

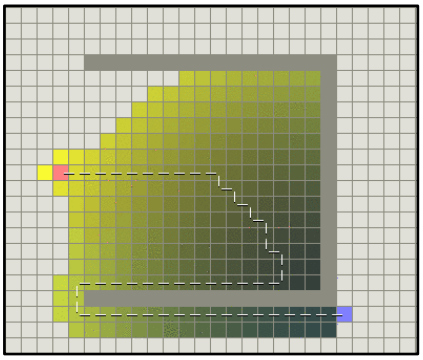


Figure 2 – The Greedy path plotting algorithm movement from the source tile (pink) to the goal tile (purple). Note that obstacle avoidance was not considered during early movements [8]

### *2.4.2 Dijkstra Search Algorithm*

The Dijkstra algorithm analyzes all possible paths to the goal and will choose the path with the lowest global cost, but the most notable compromise is the Dijkstra search process uses a significant amount of time relative to alternatives. As shown in the image below, the Dijkstra algorithm searched for a path at the top of the map when the goal was at the bottom of the map. Crucial time and resources are spent accomplishing the intended goal and evidently this process is inherently inefficient on its own.

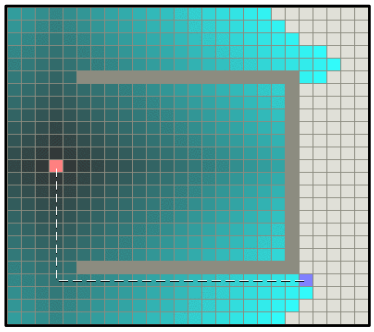


Figure 3 – The Dijkstra path plotting algorithm movement from the source tile (pink) to the goal tile (purple). Note the large number of tiles searched at the top of the grid that were ultimately insignificant to reaching the goal [8]

### *2.4.3 A Star (A*) Search Algorithm*

A star (A*) is a path plotting algorithm that plots a path the shortest distance from point A to point B. This code considers the cheapest possible and most direct route while attempting to avoid obstacles. The A* code was created by combining the Greedy Best-First-Search and Dijkstra's Algorithms, both discussed above. The Greedy code works fast to arrive to the goal but is not always efficient. The Dijkstra algorithm calculates numerous alternate paths, ultimately making it choose a better path, but at the cost of time.

The combination of both Dijkstra and Greedy algorithm gave birth to A* and as shown in the map below it combines the search method of both to make a more efficient code. A*, like Greedy, tends to search for a path in the direction of the goal, but does not move according to local cost. A* looks for the most inexpensive path to the goal like Dijkstra, but unlike Dijkstra, this code does not spend time analyzing paths unnecessary path to the goal.

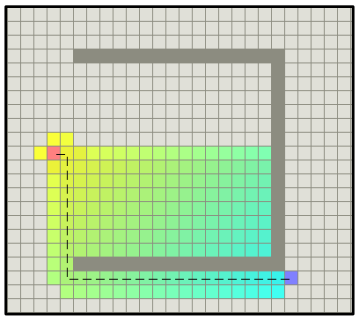


Figure 4 – A* path plotting algorithm movement from the source tile (pink) to the goal tile (purple) [8]

A* is the best combination of both of algorithms deriving its abilities form the strengths of both methods and calculates the most efficient path in the direction of the goal. A more familiar application of the A* path plotting algorithm is the implementation of A* that is used by our GPS and other path calculator programs like Google Maps and Waze. These systems rely on the A* path algorithm to plot the best routes to distant places.

### *2.4.4 Particle Swarm Drone Search Algorithm*

The particle swarm drone search algorithm was recommended to the FIU team as a feasible search method for implementation by on-campus faculty at FIU. The particle swarm drone search algorithm searches for items in an artificial map using particle swarm methods. Since it searches in an artificial environment, special consideration will be needed to ensure the search algorithm would be functional with the competition robots. Particularly, focus would need to be placed on the considerations for upgrading the movement system away from that of a particle as our terrestrial drone moves differently than that of the particle. The particle swarm drone search algorithm implements nine different search algorithms into one, namely; Sphere, Rosen, Easom, Michalewics, Rastrigin, HolderTable, Ackley, Shubert and Rosenbrock. While the article swarm drone search algorithm exhibits potential for successful search capabilities, time constraints surrounding development of related code directed the FIU team away from utilizing the discussed method.

### *2.4.5 Coordinate-Based Search Algorithm*

The coordinate-based search algorithm design consists of a custom algorithm derived by the FIU coding team and uses the location of the drone to search the field for retrieval targets, goals, and obstacles. The algorithm divides the field into designated search quadrants, then breaks down each quadrant into rows on which the rover is intended to move and search. The number of rows is dependent on the size of the search field. When the drone finds a resource, it would save the current position of resource identification in its memory and proceed to return the procured resource to the designated collection zone. The drone would then return to the point it left off before returning to search the field. The coordinate-based search algorithm showed significant potential for implementation in conjunction with additional functionalities necessary to perform the desired tasks of search and retrieval. Consequently, the coordinate-based method was the focus of FIU's proceeding with code development.

### *2.4.6 Proposed Search Algorithm*

The FIU team developed its own search algorithm. The search algorithm is a univariate linearized search algorithm that was structured using the position of the Swarmie within the expected competition area to search the field and use its location to direct it in the desired linearized pattern. The algorithm allows for linearized searches of the field to retrieve as many cubes as possible before the time limit is reached. The FIU team did not designate a scouting Swarmie to find all cube while the others pick up. Instead, all Swarmies are coded to pick up AprilCubes when identified. The algorithm has a built-in avoidance system making the robot move left or right within a row until the obstacle has been bypassed. The Swarmie will then return to the active row.

A flowchart explanation and a corresponding description of the main functions of the search and retrieve algorithm can be viewed in the Appendix of the paper.

## III. ALGORITHM PERFORMANCE IN SIMULATED ENVIRONMENT

### *3.1 Gazebo Virtual Testing Platform [6]*

The FIU team tested the chosen search algorithm with a computer-based simulation environment named Gazebo. Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex environments [6]. These characteristics, as described by *gazeboism.org*, allows Gazebo to be the ideal environment to test and optimize complex algorithms that are based in ROS, C++, and even python. The basic requirements of Gazebo are Ubuntu, a variation of Linux, as well as Nvidia video card and an Intel I5 processor. This allows the trials done to be replicated by anyone if the computer has the proper hardware.

Furthermore, with Gazebo it is possible to run virtual simulations of robots or physical robots. This allows the users to create code that can be first tested on the virtual simulation then incorporated into a physical robot to test for applications with real world stimuli. More information regarding the Gazebo virtual simulation platform can be found on Gazebo's website listed in the reference section at the end of this report.

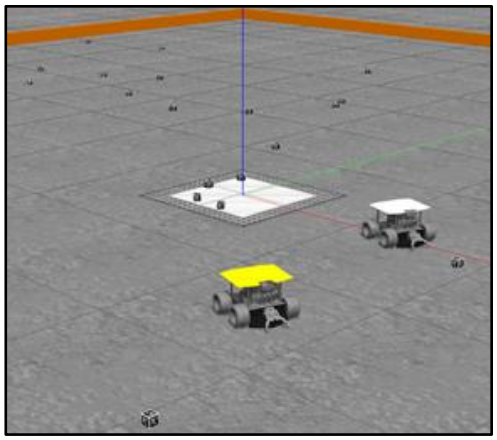


Figure 5 – FIU's Gazebo virtual simulation environment depicting two rovers, a collection zone at the center, and AprilCubes scattered throughout the simulated competition environment

### *3.2 Simulations*

The search algorithm chosen and implemented by the FIU team was initially identified as the most effective and simplest to translate given resource and time constraints. The final code developed is categorized as a univariate linearized search algorithm. The performance of this algorithm was tested against the NASA provided base code that resembles a random search pattern algorithm. The base code was found composed of a random number generator commanding the Swarmie robot to move in a randomized pattern with no goal or aim of sweeping a distinct area or path.

Virtual testing and simulations were set up in the Gazebo environment discussed above. Conducting testing in the virtual platform would provide a reliable environment capable of simulating the Swarmies at Kennedy Space Center competition field. The largest consideration regarding the subject virtual simulations are that any simulations conducted in Gazebo represent optimal conditions compared to testing perform on the physical rovers.

Optimal considerations include that the onboard instrumentation is performing as expected and no anomalies are encountered in the virtual environment. Comparing the two codes and search algorithms in this manner would allow for assessment of the performance gained over the base code without the need to consider physical robotic hardware performance at the simulation phase of the code development. In order to analyze the performance of the chosen search algorithm, the algorithm was translated into C-plus-plus (C++ or Cpp) code in the search controller behaviors, and then tested against the random search algorithm provided by NASA as a starting point.

The testing parameters were as follow: 10-minute simulations were run with three virtual rovers on a 15-meter by 15-meter field. Uniform and clustered configurations of AprilCube placement were utilized with the number of cubes increased over three runs. The testing was done with 32 and 64 resources as this proved to be the best number of AprilCubes allowed for the simulation to run without errors. The rovers started from the same position with each new trial and the cubes were placed in the exact same locations for every consecutive trial run to maintain comparable consistency. Using the plotted data from the GPS and odometer on the rqt-interface, the total area covered by each rover was summed and recorded. Since the goal of the algorithm is to identify and retrieve, and each rover is capable of recognizing multiple cubes and tagging their location, the greater the amount area covered in the allotted time yielded better algorithm performance.

### *3.3 Simulation Results*

Through multiple runs on the Gazebo virtual simulation environment, including the variations of multiple setups discussed above which include the random search algorithm and the univariate linearized search algorithm, the performance of FIU's developed algorithm over the provided random search algorithm became evident. The more linearized search algorithm developed was capable of searched a wider area in the trial defined amount of time, when compared to the random search algorithm that provided by NASA, in all but one instance. The results of the field area covered by three Swarmie robots with two types of arrangements for the cube position, and a varying number of cubes are presented in Table 1. The below table represents the four scenarios set up that performed in the simulation.

Table 1 – Search algorithm performance outcomes

| | Uniform – 32 | Uniform – 64 | Clustered – 32 | Clustered – 64 |
|---|---|---|---|---|
| Random Search | $5.85\ m^2$ | $16.23m^2$ | $8.24m^2$ | $12.16m^2$ |
| Univariate Linear Search | $8.33m^2$ | $17.08m^2$ | $7.86m^2$ | $19.48m^2$ |

According to the outcomes represented in Table 1, the univariate linearized algorithm allowed for the rovers to each cover a higher amount of area in the allotted 10 minutes in all virtual simulations excerpt for the Clustered-32 trial. Based on the virtual simulations, the FIU team choose to utilize the univariate search algorithm developed. This algorithm showed the most promise considering additional functionality operations in need of development for

successful physical testing. Of the additional operations, the pickup controller function is among the most pertinent for successful retrieval of identified resources.

## IV. ROBOTIC INTEGRATION

### *4.1 The NASA Swarm Robot – "Swarmie"*

The first task regarding integration to the physical platform undertaken was analyzing the provided Swarmie. Gaining a foundational understanding of how the individual components of the robot are laid out and function aided in code development and ensured the provided on-board equipment of the robotic system was undamaged and would function properly. Each Swarmie is equipped with ultrasonic sensors, a live-feed camera, a GPS, a compass, wheel encoders, a gripper claw assembly, and an inertial measurement unit. The Swarmie was assembled according the instructions provided by NASA and available on the competition website.

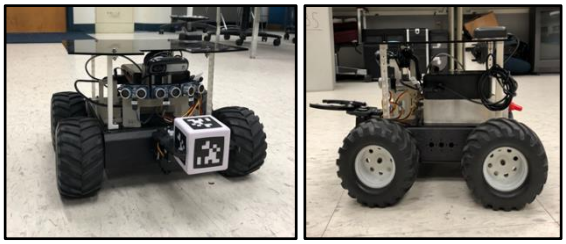


Figure 6 – Front and left side views of the assembled Swarmie provided by NASA

### *4.2 Computer Vision and Open CV*

Computer vision is a multidisciplinary field pertaining to computer capabilities for developing a powerful level of understanding from images and videos. OpenCV is an open source computer vision library equipped with hundreds of functions and algorithms for processing images and videos with support in C++ [5]. The OpenCV platform is utilized in the Swarmathon as means of recognizing the AprilTags on the AprilCubes and the collection zone or home base. OpenCV functions and algorithms use the live feed from the Logitech C170 camera mounted on each Swarmie. The camera feed is dependent on the limitations of this specific camera, as well as outside variables such as lighting and stability while the rover is in motion. The FIU team focused on exploitation of the provided code to merge an efficient algorithm with the OpenCV capabilities.

The on-board camera is the eye of the robot and it is used to detect and recognize objects, specifically AprilTags and subsequently AprilCubes. The camera used in the Swarmie is pointing towards the ground at a 30-degrees from horizontal and works with internal software to detect and identify the AprilTags. This software was coded to detect a target at 0.65-meters in front of the robot's gripper assembly. The figure above shows the relationships between the AprilCube and the Swarmie, and how the distance values to each were calculated. This calculation is based on measurements of the hypotenuse and adjacent, using the distance of target the camera detects as the hypotenuse and the adjacent is known as it is a physically measurable distance. The system works with OpenCV and Computer Vision to identify the AprilCubes and is an integral portion of the overall success related to resource retrieval performance.

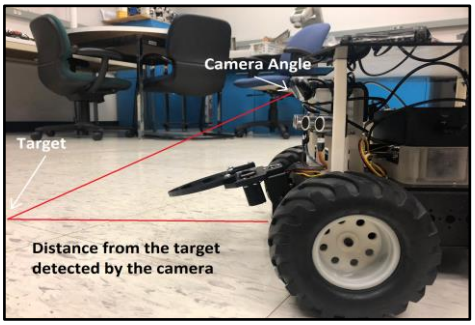



Figure 7: Camera and target identification with reference geometry depicted as red lines

### *4.3 GPS - Global Positioning System*

A low-power consumption GPS chip can be found on board of each Swarmie. NASA has equipped them with the LEA-6 modules, capable of high performance per the u-blox 6-position engine. The GPS utilizes the doppler shift in radio frequencies in determining lines of both position and location. Using triangulation from the information of at least 3 satellites, signals with time stamps are collected by the GPS and compared to the time it arrived in order to produce longitude and latitude coordinates. GPS bias is usually remedied by allowing clock errors as a variable, allowing the receiver to successfully locate the Swarmies position on Earth.

### *4.4 Inertial Measurement Unit (IMU)*

The IMU unit on each Swarmie reports x, y, and z acceleration of the robot, as well as orientation using the magnetic north as reference. Exploitation of the onboard IMU will produce data that can be checked against other instrumentation data. The IMU data will also help limit the Swarmies' speeds to values within the competition rules.

### *4.5 Kalman Filters*

The collected data is checked through offset calibrations, which were physical values that can be double checked with the code for accuracy, and through the use of Kalman Filters. Kalman Filters are an optimal recursive data processing algorithm pertinent to continued function of the algorithm. The filter is capable of analyzing all the provided input from different sensors and returns optimized data that parallels the system's needs. A graphical representation of the application for a Kalman Filter is as per the figure below.

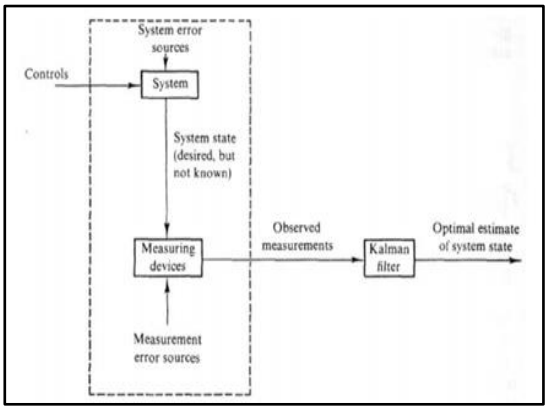


Figure 8 - Common Kalman Filter Application

# V. METHODS

## *5.1 Global Positioning System (GPS)*

The GPS was given extensive consideration while developing the Swarmies' code. The location reference the GPS provided was essential in the design of the pattern the Swarmies will be taking. The GPS must take into account the location of the Swarmies to accurately map the patterns in which the other Swarmies move and where obstacles and cubes are located. The GPS is primarily utilized by calling a specified function to execute do-while loops that allows the Swarmie to move to locations using its GPS data.

## *5.2 Encoders*

The FIU team implemented minimal changes to the provided wheel encoder program. Past FIU's teams performed in-depth evaluation of the encoders related to the same design goals and provided information stating that the encoders were not an accurate information source. Reliance on the encoders made the robot perform incorrectly and produced excessive error when relying on the wheel encoder sensors for navigation.

## *5.3 Compass*

The compass was implemented to work with the GPS and the search algorithm to calculate the current direction the robot is moving. The compass proved a vital tool for establishing pertinent information related to the Swarmie's heading data.

## *5.4 AprilCube Pickup and Drop Off*

The FIU team changed the pickup and drop-off code to work on a distance-based calculation and continuously used the GPS and odometer to calculate the rover's current position while the camera was used to update the AprilCube position. The original pickup and drop-off sequence installed on the robot relied on a time-based countdown and control sequence where it would perform different actions based on how much time had passed. This method proved ineffective and failed during initial testing and was quickly abandoned for the method described above.

The new method ensured the Swarmie was able to correctly identify and collect the targeted resource, and then not drop or lose control of the AprilCube on its way back to the collection zone. If dropped, the robotic code was constructed to tell the Swarmie to move back and search for the AprilCube in a 160- degree range from the last known location.

# VI. PHYSICAL TESTING

## *6.1. Initial Configuration and Testing*

Testing was conducted to establish and understand the Swarmies' capabilities and functionality after the search algorithm was developed and tested on the virtual platform. The testing of the Swarmies was conducted in FIU's robotics lab in both the provided simulated environment using Gazebo as discussed above, and a mock course built in the robotics lab to execute physical tests and evaluate the results as discrepancies were anticipated between the virtual and physical simulations.

The model course included the NASA specified home base, and replicas of the competition AprilCubes that featured printed AprilTags on six sides of the cube. Initial configuration of the Swarmie contained parameters that were interfering with other operations during the starting procedures. To resolve this, the FIU team conducted individual component tests or evaluations to check how sensitive each component was. The observable data was compared with the output data displayed on the rover rqt-interface. The rqt interface proved to be a crucial tool for evaluating physical performance. See the Figure 9 depicting a screenshot of the rqt-interface.

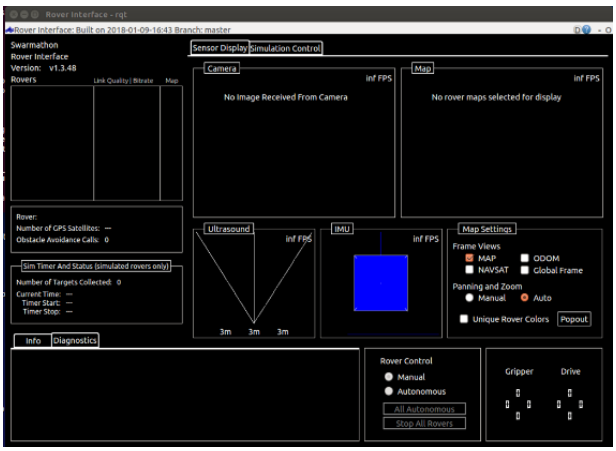


Figure 9 – Rqt-interface or rover interface

## *6.2 Calibration*

Calibrations were performed for each component to increase accuracy. The calibration was done initially by testing the rover's functions related to the onboard camera, GPS, and gripper functionality. To specifically determine the offset of the actual Swarmie, the distance was measured from the chassis to various points of the gripper, the chassis to the camera, the camera to the grippers and ground, various range of angles the camera could function, as well as the actual velocities that the Swarmie travelled. Appropriate comparisons were made between physical and code produced data. The FIU team placed AprilCubes at various distances and angles away from the rover. Camera and ultrasound

sensors stated values were recorded and compared to the actual known distances to determine at what values the camera and ultrasound should respond to and operate based on.

### *6.3 GPS*

GPS testing was done by taking screenshots of the Rover interface (rqt-interface) displaying NAVSAT, ODOM, and position data produced by the physical rover during simulations. The position of the Swarmie was then changed after each trial and the testing was repeated. The results of all tests were compared to assess the accuracy and reliance of the GPS. While testing the GPS, the FIU team noticed that the rover's reported location was not as accurate as initially anticipated. Further investigation led to the conclusion that the GPS hardware and software the robotic rovers utilize can produce up to a 5-foot margin of error.

### *6.4 Compass*

Testing the compass was conducted by rotating the Swarmie while comparing the direction the Swarmie was reporting though the rqt-interface with an actual compass in hand inside the robotics lab. The complete testing was done using 20-degree directional change increments and the results were compared. No significant discrepancies worth offsetting was noted during the compass evaluation.

### *6.5 Wheel Encoders*

The wheel encoders were tested by measuring and marking the wheels of the Swarmies, then commanding it to move in a desired direction. Through the rqt-interface, the information provided by the operating Swarmie was noted and then compared with the known physical data information.

### *6.6 Ultrasound Sensors*

Testing for the ultrasound sensors was done by placing an object in front of the Swarmies within recognizable distance to the sensor's mounted location. A measurement was taken by hand to compare with the produced data representing the distance between the sensor and the object.

## VII. RESULTS

### *7.1 GPS*

The position data taken by the algorithm is usually error prone, as each iteration of the GPS would locate the Swarmie at different positions within a specific range as the data is converted to x and y-coordinates on the world map coordinate. This can be appreciated by the rqt GUI interface map display as per the Figure 10, where the dots are GPS x and y-coordinates, and they propagate at a certain distance from the IMU and wheel encoders data.

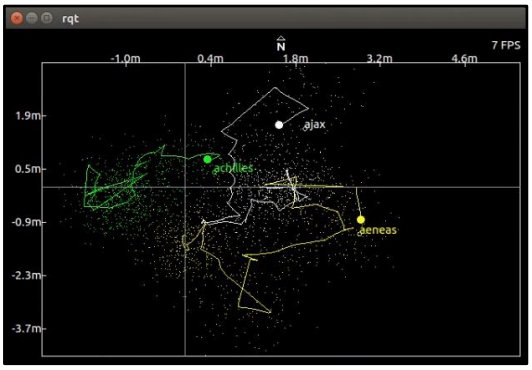


Figure 10 - IMU and GPS Graphical Data

### *7.2 Ultrasound Sensors*

While testing the ultrasound sensors, The FIU team observed the calibration of the sensor was initially too sensitive and was finding objects from 5-feet away. The increased sensitivity caused the Swarmie to enter an evade mode as though it encountered an obstacle to avoid even though no obstacle existed.

The ultrasound sensors were adjusted to a lower sensitivity that resulted in the Swarmie operating more predictably and as coded. The Swarmie no longer entered the evade mode after the adjustments were made. The adjustments subsequently allowed the Swarmie to identify and pickup AprilCubes as intended. The ultrasound sensitivity adjustment enabled proper function of the Swarmie and behavior as intended by the FIU coding team.

### *7.3 Wheel Encoders*

It was concluded that noise picked up by the IMU and GPS, and an amount drift accumulated by the wheel encoders, facilitated unpredictable divergence from the intended path and caused the Swarmie to become lost or simply have inaccurate position data. The noise captured can be appreciated in the following figure, part of the rqt and GUI interface. The amount of noise and drift is displayed while running simulations and running code on the physical test Swarmies.

Figure 11 - GPS and IMU GUI offset data

The data gathered by the GPS, IMU, and Wheel Encoders was checked against each other and through the EKF (Extended Kalman Filter) and results showed that after filtering the data nearly represented the correct position of the Swarmie. The Extended Kalman Filter is a built-in option that FIU implemented at discretion. The FIU team decided to apply this utilized Kalman Filter due to its capacity to linearize.

### *7.4 Camera*

Although the image quality of the camera was an initial concern while the Swarmie was in motion, laboratory testing concluded this did not impede the Swarmie's search and retrieve process. The FIU team was satisfied with the camera's functionality after sufficient testing proved the images produced during slower movements were reliable and allowed for proper identification of AprilTags.

### *7.5 AprilCube Pickup and Drop Off*

The adjustment from a time-based pickup to a distance-based pickup was an invaluable change implemented by the FIU team. When the Swarmie positively identified an AprilCube, it successfully completed the pickup function approximately 70% of the time on its first attempt and was observed to take a maximum of three attempts to retrieve the AprilCube as intended. Adjustments to the pickup and drop-off function over the provided base code resulted in better overall Swarmie performance.

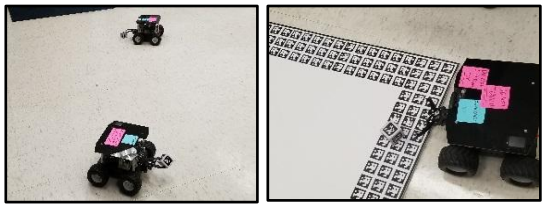


Figure 12 – Swarmies executing pickup and drop off functions during laboratory testing

## VIII. CONCLUSION

The development of FIU's implemented algorithm was a direct outcome of the physical constraints found through multiple testing iterations and time allowance. The team observed that the calibration of the rovers' sensors was extremely important in the physical test as the results from the performance of the virtual simulations varied from the real-world testing of the rovers, as expected, as the Gazebo platform offers ideal testing conditions. Ultimately, the comparison of the base code provided by NASA and FIU's univariate linearized search algorithm revealed improvements were made over the base code. The outcomes were sufficient to move forward with code development.

FIU's final approach to the search algorithm drew inspiration from two main sources, a biological approach derived from ant colonies' behaviors and the linear motion of chess pieces on the game board. Using linearity, the rovers are coded to search all sections of a determined area as they attempt to move in straight patterns to cover the entire developed grid. This search algorithm was tuned through the rqt-interface as the rovers searched for the AprilTag resources in the laboratory test environment.

Obstacle avoidance allows for the rovers to cover the entire grid and camera vision differentiates between an obstacle or AprilTag. The GPS and odometry package played an important role in improving the ability to retrieve AprilCubes and successfully place them in the home base. The Kalman Filters allowed for more reliable data from the GPS. The use of better performing hardware could increase the efficiency of the algorithm's performance. FIU's algorithm and improvements on the pickup control may be used in later editions of FIU teams participating in the NASA competition.

A flowchart explanation and a corresponding description of the main functions of the search and retrieve algorithm developed can be viewed in the Appendix.

## IX. ACKNOWLEDGMENTS

The construction and integration of the FIU team's search and retrieval algorithm was made possible by the Engineering Manufacturing Center of FIU who offered unwavering support throughout the development of the subject research. A special thank you to the coding team, particularly Naadir Kirlew, for devoting a significant amount of personal time to code development. Thank you to all those who contributed to research and development of the subject algorithm through extensive research and coding including but not limited to: Kumar Yogesh Shah, Grant Wilcox, Daniel Manfred Klumpp, Steven Andrew Garcia, Nelvin Chery, Ray Nicolas Santamaria, Mellony Marie Ladino, Rami Ghazzara, Luyan Zhang, Jose Miguel Tormo, Pietro Dimitri Gomez, Carlos Celemin, and Abdulaziz Alenezy.

## X. REFERENCES

[1] Altshuler, Yaniv, et al. "Swarm Intelligence — Searchers, Cleaners and Hunters." Unknown Publisher and Date.

[2] Mohan, Yogeswaran, and S.G. Ponnambalam. "An Extensive Review of Research in Swarm Robotics." ResearchGate, Jan. 2010.

[3] Moore, Thomas, and Daniel Stouch. "A Generalized Extended Kalman Filter Implementation for the Robot Operating System." SpringerLink, Springer, Cham, 3 Sept. 2015.

[4] Olson, Edwin. "AprilTag: A Robust and Flexible Visual Fiducial System." April Robotics Laboratory, 2011

[5] "OpenCV Library." OpenCV Library, OpenCV Team, 2018 [Online]. http://opencv.org/. [Accessed Jan-Mar 2018].

[6] Osrf. "Beginner: Overview." Gazebo, Open Source Robotics Foundation, 2014, gazebosim.org/tutorials?tut=guided_ b1&cat

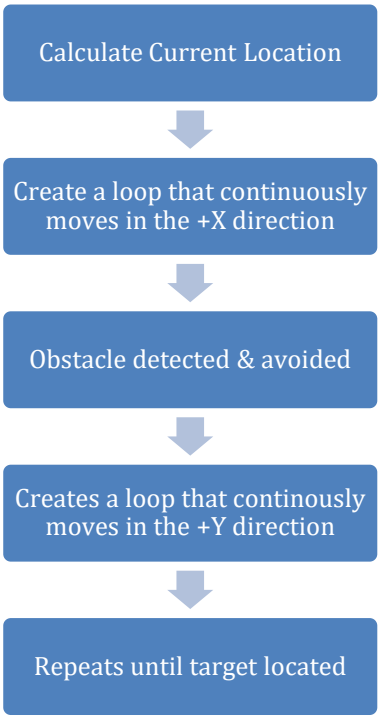
[7] Parker, Lynne E. "Distributed Intelligence: Overview of the Field and Its Application in Multi-Robot Systems." Association for the Advancement of Artificial Intelligence, 2007.

[8] Patel, Amit. "Introduction to A-Star (A*)." Introduction to A*,1997,theory.stanford.edu/~amitp/GameProgramming/A StarComparison.html

[9] ROS Wiki, "ROS/Introduction," Wiki.ros.org, 2018. [Online]. http://wiki.ros.org/ROS/Introduction. [Accessed Jan-Mar 2018].

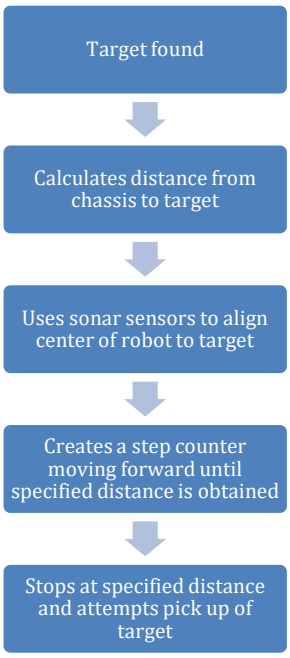
## APPENDIX: FINAL CODE EXCERPTS AND DESCRIPTIONS

**Search Controller Flowchart:**



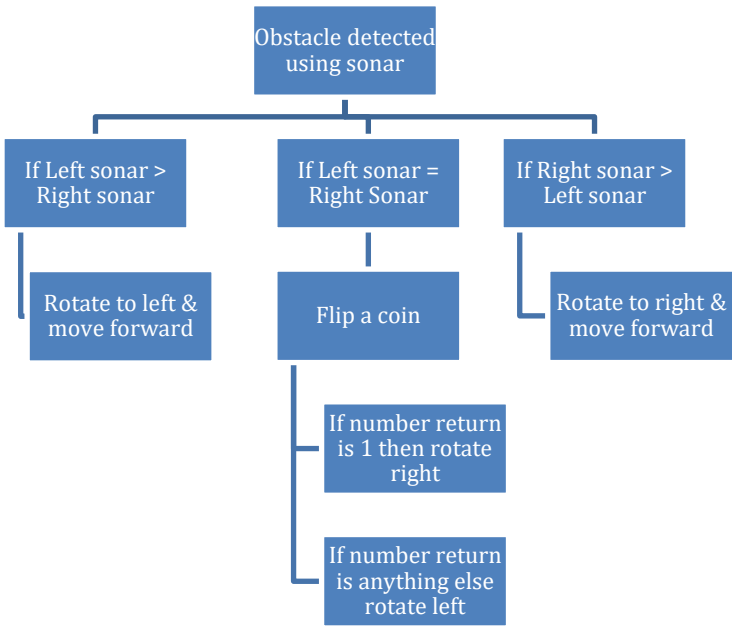

The flowchart representation of the FIU code shown above depicts the search algorithm. The concept behind the search algorithm was derived from the game of chess. The code programs the robots to create a grid based on search area. In the scenarios that were tested, the robots were already programmed to know what the grid size was and coincided with the competition grid. The robots were then able to move in a linear pattern that, between three rovers, would ensure the majority of the terrain was searched at least once. As such, the robot would first use the GPS to understand what it's current location was and break the coordinates down into x and y values. The robot then was programed to move in the positive x-direction by creating a loop that would continuously move forward until an obstacle is detected and then would change directions and move in the positive y-direction. This process was repeated and tested numerous times and was determined that the robots were able to search the majority of the terrain as intended.

**Pickup Controller Flowchart:**




Above, the flowchart represents the FIU code used to retrieve a cube as shown. There were numerous methods tested including time-dependent, position-dependent, sonar-dependent, and camera dependent. Ultimately, the code that worked the best and was used in the final application was one that combined position, sonar and camera to ensure successful retrieval. Once a cube or resource was identified, the robot would calculate the position of the cube using the hypotonus from the camera and the center of the chassis. Then the rover would create a loop recalculating the distance of the cube from the robot until a predetermined distance was reached. In this scenario, that distance was approximately 3.75-inches, which was the ideal location of the gripper to successfully retrieve the cube. In order to ensure the robot stayed perfectly aligned to the cube, ultrasound was incorporated. Three sonar sensors were mounted above the gripper and below the camera allowing for the rover to detect objects to its left, center, and right. As such, the code was made so that once a cube was identified the robot would rotate either left or right so that the center sonar was the only one "blocked" indicating that the cube was directly in front of it.

**Obstacle Avoidance Flowchart:**




The flowchart shown above represents the FIU code implemented for obstacle avoidance. In order to ensure the robots were capable of successfully navigating terrain that included obstacles, an algorithm was written named Obstacle avoidance. The algorithm was based primarily on the sonar sensors. The sonar sensors were capable of detecting objects farther away than the camera's visibility could discern. In addition, since three sonar sensors were used the robots were also able to detect which direction the obstacle was located at and more accordingly. For example, if an obstacle was detected in the left and center sonar while the right sensor was clear, then the robot would rotate towards the right allowing the robot to clear the obstacle without difficulty. Similarly, the robot would rotate to the left if the opposite scenario was presented. In the event that both the right, left, and center sensor detected an obstacle, the robot would "flip a coin" and then assign a value to heads and tail and move either right or left depend on the result.