ENAE788M - The Final Race

Team `sudo rm -rf *`

Abhishek Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: shastry@umd.edu

Animesh Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: animeshs@umd.edu

Nicholas Rehm
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: nrehm@umd.edu

***Abstract*—In this report, all the missions including wall avoidance, gate detection, bridge detection, and target detection are implemented within a single flight for ENAE788M: Hands on Autonomous Aerial Robotics. Minor improvements have been made to each mission to improve overall system robustness. The framework for each mission includes the required detection algorithm as well as its corresponding mission node which generates the required flight maneuvers for each task. These ROS nodes' primary function executions are managed by a mission index topic, which is only updated upon completion of the current active mission. This method allows all nodes to be running at system startup without exceeding hardware limits which in turn saves time throughout the flight because nodes do not need to be shut down and started while in flight. This system has been optimized for both robustness and speed to achieve roughly 95% success rate with completion times just under 1 minute.**

I. Introduction

A quadrotor navigating in a partially known environment must be able to adapt to various obstacles of uncertain position and orientation in space. These obstacles may include walls, windows, and ground targets. The benefit of a partially known environment is that the order of obstacles is known ahead of time so an advanced state machine is not required to handle complex decision making onboard. Rather, a simpler approach can be employed in which the completion of a task triggers the start of the next task.
Link to the result videos: Click Here

II. Sub-mission Optimization

Each sub-mission has been tuned from previous projects in order to provide the best performance in the context of the final race. General approaches and changes to each task are listed here:

A. Wall Detection and Navigation

Wall detection is achieved by spatially matching features through our method of "constructing" our own camera baseline through the motion of the quadrotor. Depth of the matched features is computed and the extrema of the features with similar depth are used to estimate the boundaries of the wall. Due to the wall heights in the arena being known, a very strong bias in the position and height of each wall can be used to ensure consistent detection. In practice, just one measurement is enough to avoid the wall.

B. Gate Detection and Navigation

Gate detection is achieved through a simple color threshholding in OpenCV as it was found that proper GMM implementation was too computationally costly. A convex polygon is fit to the largest contour in the threshholded binary image and the corners of that polygon are used to solve for the pose of the window using solvePnP implemented in OpenCV. The estimated position of the window is filtered using an EKF, which only allows the quadrotor to traverse through it after it has properly aligned with the window and the co-variance has reduced within our pre-defined limit.

C. Bridge Detection and Navigation

Textured areas of the downward facing camera are identified by the presence of edges using canny edge detector in OpenCV. Of the untextured areas, a simple brightness threshholding is used to identify the bridge, which is brighter in color than the untextured river or black foam floor mats. The proportions of the resulting bridge mask are used to find the bridge orientation which is used in computing waypoints before and after the bridge for the quadrotor to navigate to.

D. Target Detection and Navigation

To detect the circular target, the downward facing camera is first threshholded for brightness to isolate the white poster that the target is printed on. Circles are then fit to this image. The average of their centers is taken to find the location of the target within the frame, and the average of their sizes is taken to find the equivalent size of the target which is used for depth estimation. Target position is filtered using an EKF which only allows the quadrotor to land on the target when the co-variance of the estimate has reduced within our pre-defined limit. It was found that the exact same method can be used for the square target with slight tuning to the circle fitting parameters. In doing this, the total number of nodes running on the vehicle was reduced.

E. Controller

We use a simple proportional derivative control law using the Bebop's onboard odometry as feedback. Waypoints are published to the controller as reference commands. An error is then generated and its magnitude is saturated upon crossing

a desired threshold before transforming it into the desired body velocity commands by the PD control law. The saturation is done primarily to improve the odometry feedback, as faster motions result in inaccurate flow and sonar measurements. Also, the vision is more accurate if the vehicle is close to hover state. It should be noted that the Bebop driver publishes odometry at a rate of 5 Hz which is the limiting factor in terms of performance.

III. Implementation Approach

A. System Architecture Overview

The complete software for vision, high level decision, and control runs on the Intel UP Board mounted on the Bebop as an onboard processor. The overall system consists of 3 primary layers of nodes and a single topic that manages when a particular set of nodes are executed. The outer layer is the detection layer: the nodes inside it take in the raw images from the front monocular camera and the bottom stereo camera. It processes the images and detects the object of interest corresponding to the active mission. Pose information is then obtained from the detected object, which is then transformed into inertial coordinates and sent to the middle layer. Inside it, the corresponding EKF node takes in the raw poses and filters out the unreliable ones. The filtered, or trustworthy, pose information of the object of interest is then passed on to the inner layer, comprised of mission nodes that generate the appropriate reference poses for the controller, so that the active mission is successfully completed. Only the nodes in this layer have the access to change the data in the "Mission Index" topic, and consequently decides when to kill itself and activate the next mission.

B. General Mission Structure

All of the missions, like flying over the wall or through a window, or landing over a target have the same sequential structure. The mission either starts with a take-off command or with the completion of the previous mission. Next, the mission follows a three step process that generates appropriate reference signals for the controller in the form of desired pose. The transition from search to converge step occurs on the first reliable detection of the object. The action of moving to the waypoint before the object is denoted the converge step, and upon moving within acceptable bounds on position and velocity, the next step, called the execute step, effectively generates a waypoint located beyond the object. After the completion of the execute step, the mission either ends itself and calls the next mission or lands the quadrotor.

C. General EKF Structure

A standard EKF is used as an observer in most cases, but here it is used to filter out the false detections provided by the vision algorithms. A reliable detection can be defined as one where a series of detections happen within a short period of time. For example 10 detections in 1 second can be safely considered reliable. On the other hand, detections that happen very rarely cannot be trusted. Hence, the EKF, by the virtue of having covariance as a part of its state, is used here to measure the detection reliability and to supply only the trustworthy measurements to the mission node.

D. Managing CPU Load

Launching the corresponding nodes using a script or sub-process or service calls is a very convenient method, but it requires the RAM to be cleared and loaded up at each mission switch in mid-flight. On, the other hand launching all the nodes at once will save a few seconds but the CPU will be overloaded and the system will crash. To combat the CPU overload, a ROS topic that stores the mission index was created. All, the running nodes can read this mission index and if it corresponds to its own index, then the primary function which is computationally heavy is executed. In summary, assuming that RAM is enough, all the nodes are launched at once, but their primary functions are only called when it's their time. In, this way, the CPU load is controlled and a few seconds of loading and unloading the RAM mid-flight is avoided, at the cost of heavy RAM usage and programming complexity. Fig1 shows CPU usage of Upboard with time.

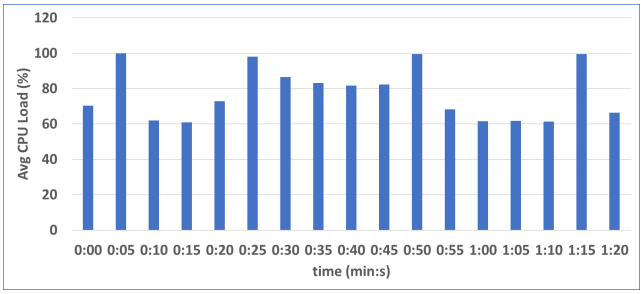

Fig. 1. Average CPU load with time for the entire course

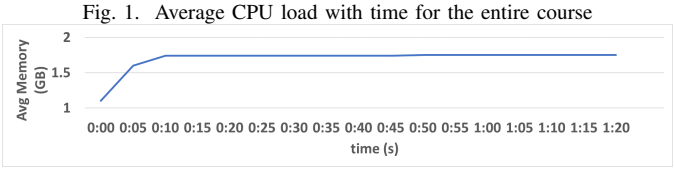

Fig. 2. Average RAM usage with time for the entire course

IV. Results and Conclusion

Our method of completing the final race in which sub-tasks are managed by a master mission node that only allows for sequential execution of tasks was successful in completing the obstacle course with a fast time. With conservative control gains and waypoint convergence criteria, our method can consistently complete the course in roughly 1 minute and 15 seconds. With more aggressive control gains and loose waypoint convergence criteria, the completion time was reduced to just under 1 minute. With all of the sub-mission tasks tuned for performance, the limiting factor in our method's speed becomes the reliance on waypoints for our main control scheme. To increase performance speed, this system would have to be fully overhauled with a velocity-based control scheme. Such a method is more prone to error and crashes while testing which is why we have opted for our approach throughout this course.

V. System Architecture Overview

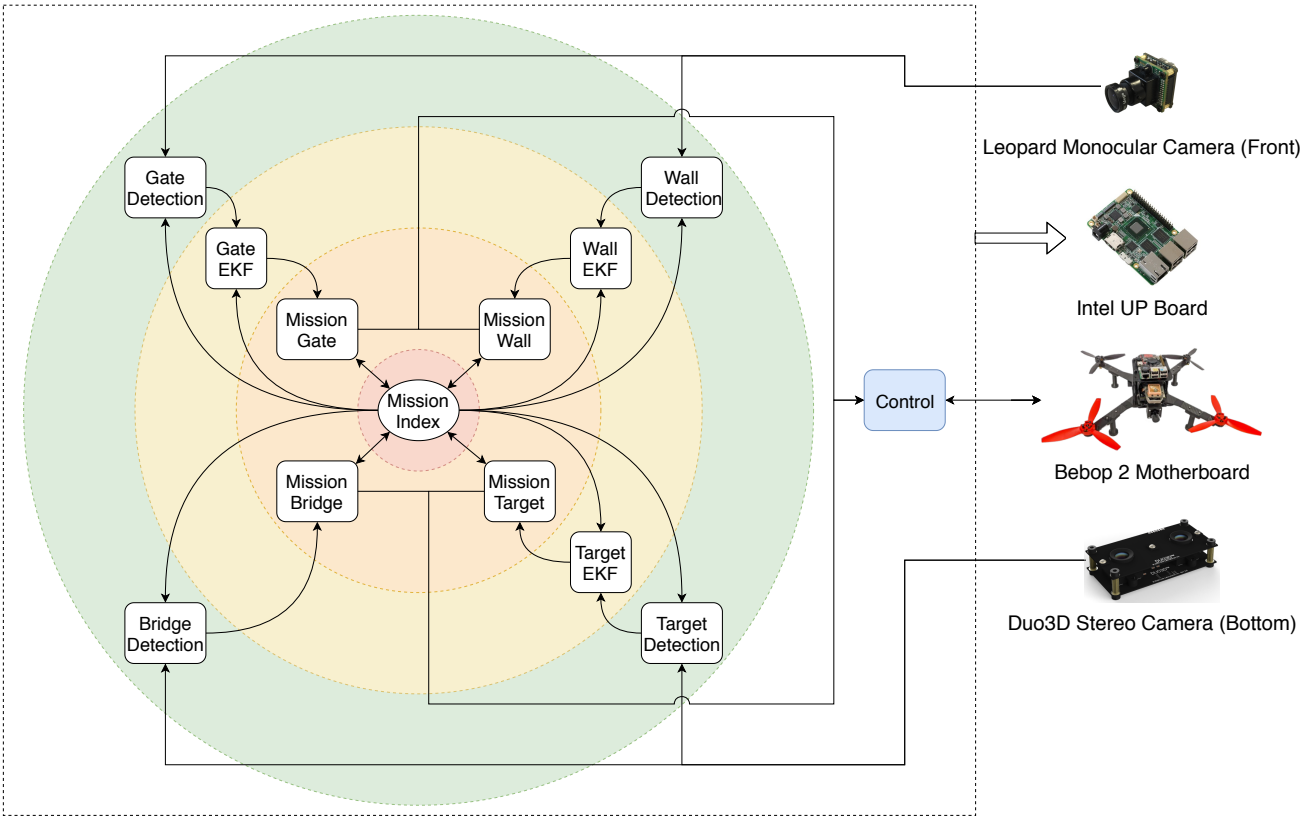

Fig. 3. Complete system architecture.

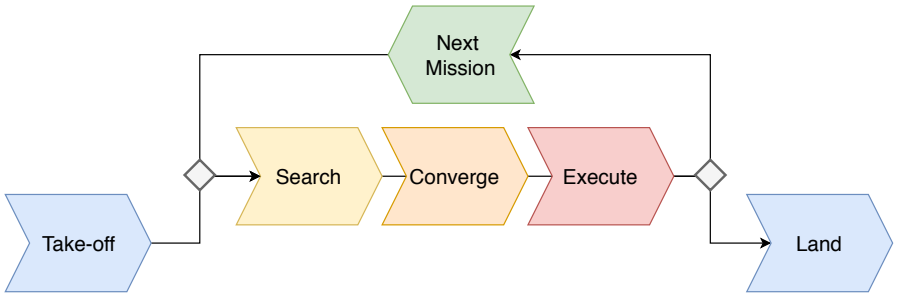

Fig. 4. General Mission Structure.

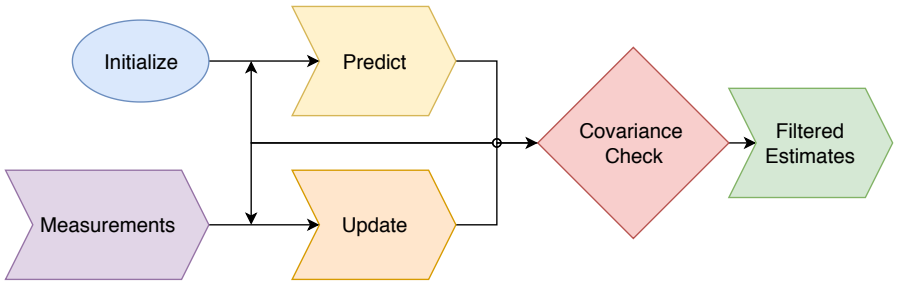

Fig. 5. General EKF Structure.

VI. Summary of all the vision algorithms

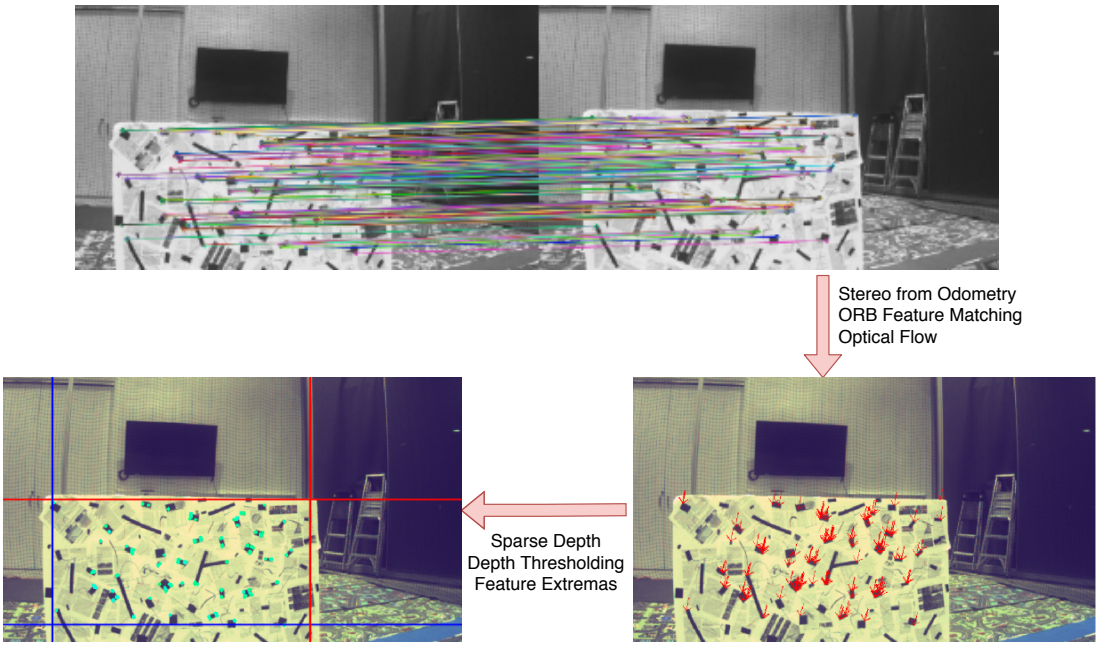

Fig. 6. Wall Detection.

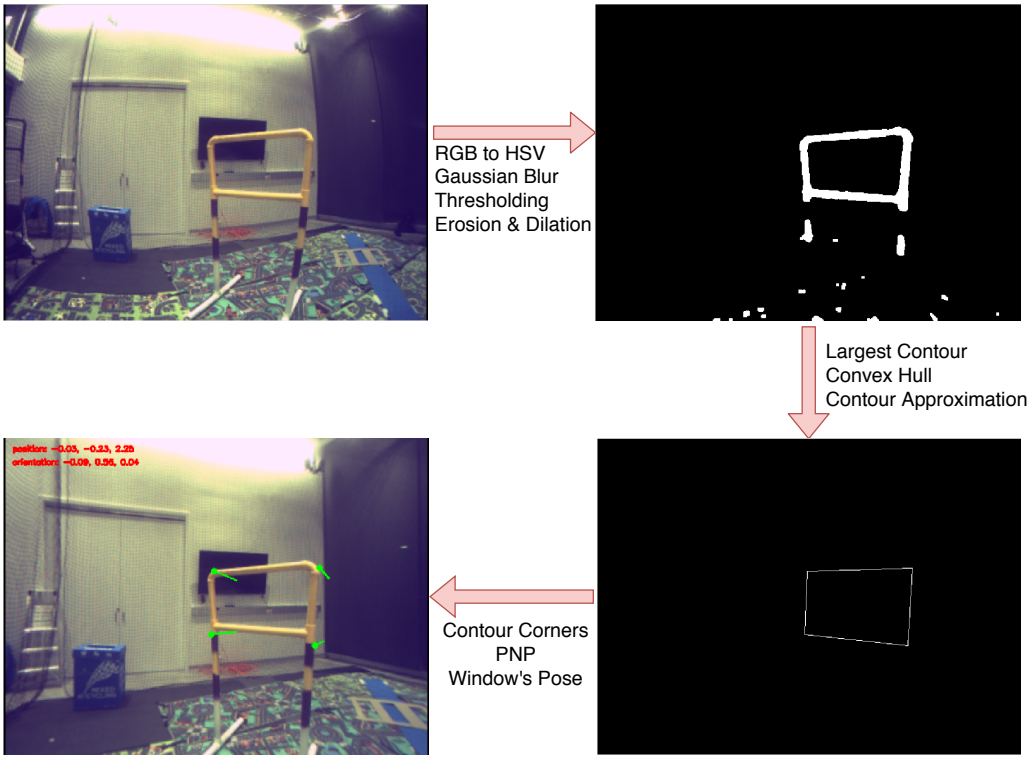

Fig. 7. Window Detection.

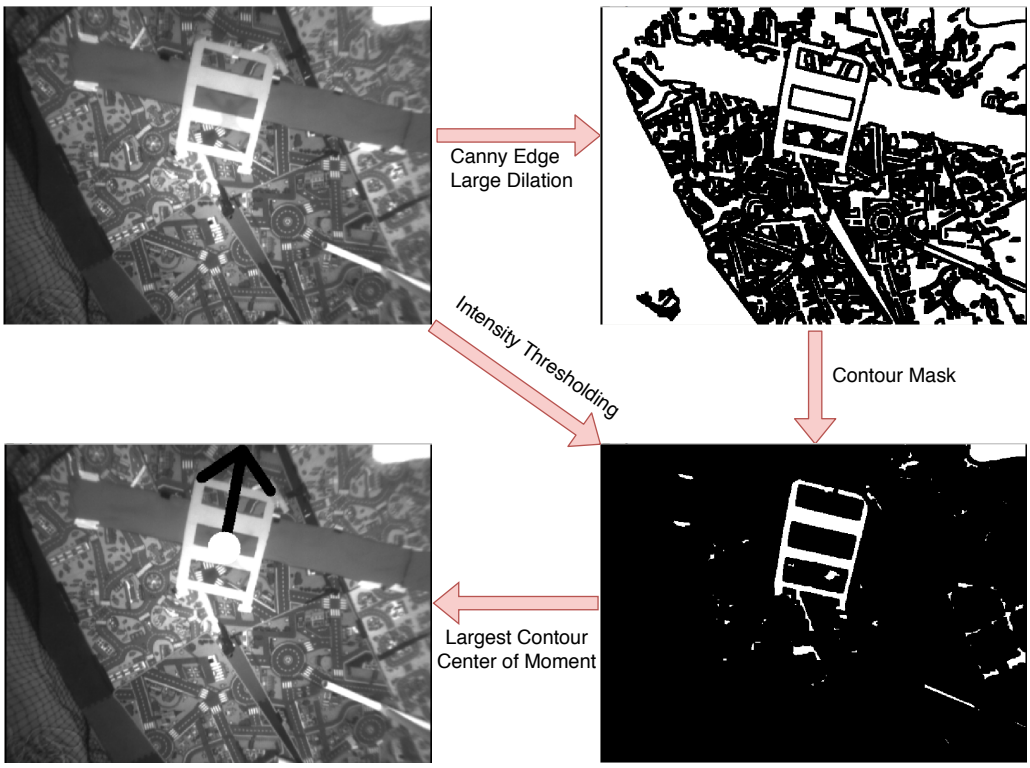

Fig. 8. Bridge Detection.

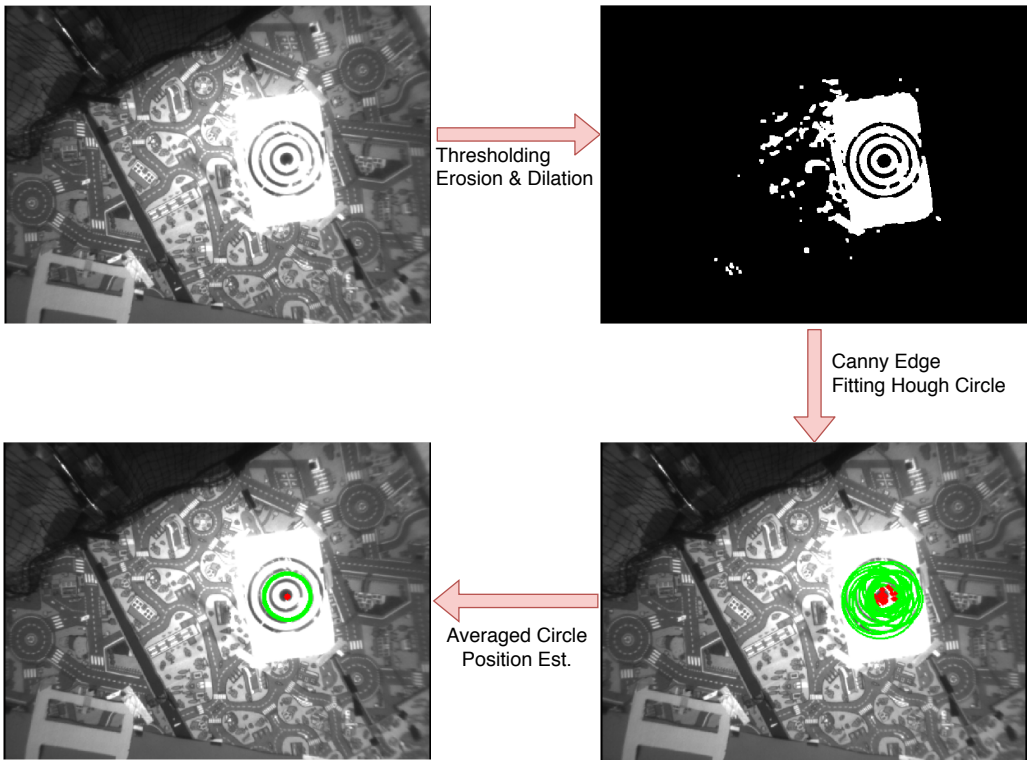

Fig. 9. Bullseye Target Detection.

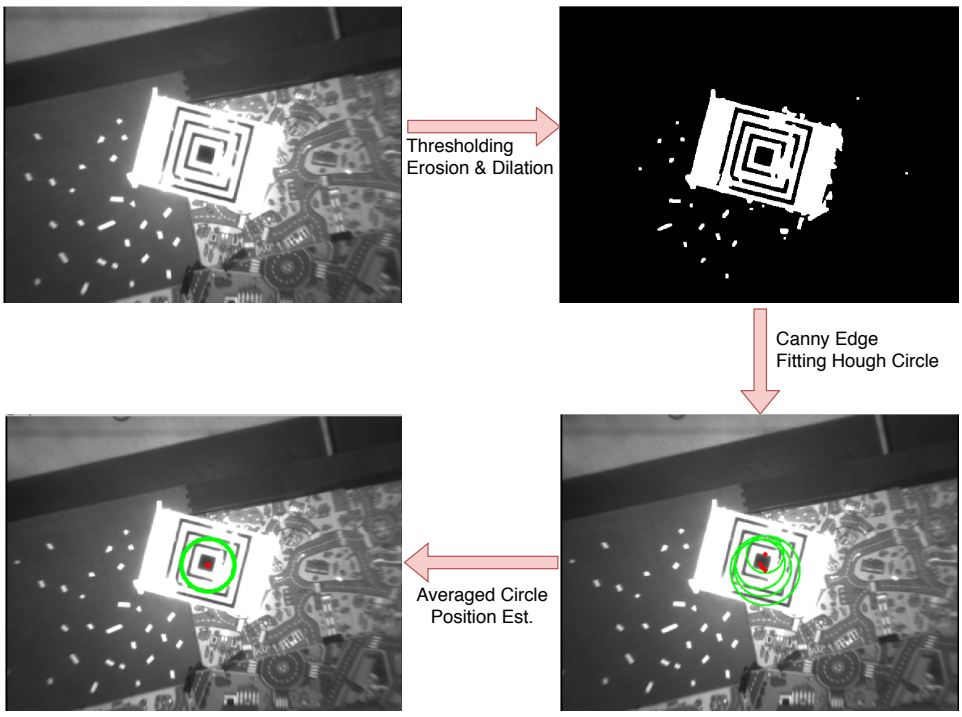

Fig. 10. Square Target Detection.