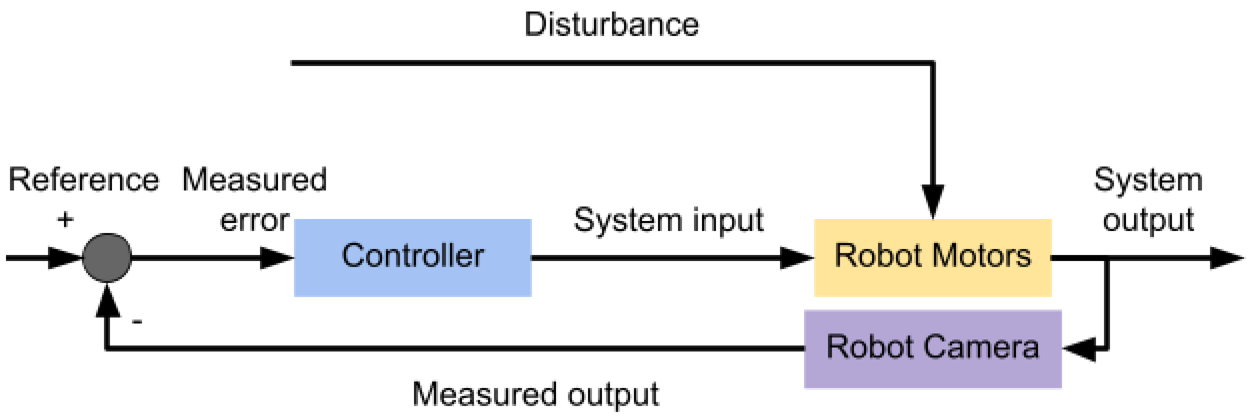
HW 2 Report

Calibration

For our calibration, we took 10 pictures of a 9x9 checkerboard with the camera on the robot using the “ros2 image view image saver” command. Using the OpenCV approach, we ran the images through the provided functions (https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html) to calculate the calibration parameters of the camera: the camera matrix (including intrinsic parameters like focal length [fx, fy] and optical centers [cx, cy]) and the distortion coefficients.

Model

(a) The description of your closed-loop controller. You should draw a similar graph like below, and give a short description of each node and the information being passed on the edges.



Our system consists of the robot motors and the robot sensor, but we split it into two nodes for clarity, since the robot camera is also the sensor in this case. Assume the robot motors node also includes the main robot body. The disturbance to the motors would be friction and other such environmental factors. In our loop, we have the controller which calculates and sends the motor commands (derived from the twist) needed to reach the next waypoint as system input to the robot motors. Since our system technically includes both the motors and camera, the system output and measured output are essentially the robot pose in the world frame (calculated from the camera measurement of the AprilTag’s pose). The measured output is then compared to the reference value (estimated pose). The difference (measured output - reference value) is the measured error that is then fed back into the controller to recalculate the twist needed to reach the next waypoint, completing our feedback loop.

(b) The description of your landmark. You need to talk about where you place landmarks, and give a short reason why you position them that way.

We used the AprilTags, positioned in the world such that the robot points toward one or more of them as much as possible during traversal. This way, we can get the most accurate traversal since it will often be correcting for error based on the actual location of the AprilTags. We specifically placed them at: (1.5,

(c) A description about how to estimate the car pose based on the landmarks and how to handle the noise in the estimate of landmark poses. You should motivate your choice of a solution.

We made the start pose of the robot (0, 0, 0) in the world frame. We calculated the car pose based on the AprilTags in two ways. First, we converted the AprilTag pose retrieved in the camera frame to the robot frame, by converting the (x, y, z) axes. Specifically, we converted the +z (forward), +y (downward), +x (to the right) for the camera to the +x (forward), +y (to the left), and +z (upward) of the robot. We did this by applying two π/2 rotations using 3D rotation matrices R_z and R_x. This allowed us to get the (x, y) coordinates of the robot in the world frame, since the world frame is +x (to the right), +y (forward), and +z (upward). Finally, we used the y-orientation of the AprilTag pose subtracted from the known orientation of the AprilTag in world frame to determine the orientation of the robot in the world frame. (FIX)

Ultimately, we had very little noise in our AprilTag pose estimates, but we accounted for noise by averaging the estimated pose of the robot in the world frame based on multiple tags. This way, our robot could be less impacted if the estimated pose of one AprilTag was off.

(d) During motion, in some short time, the camera may not detect any landmark. You must talk about how you handle the case when there is no landmark in the camera field of view or you can't detect any landmark.

When there is no landmark in the camera field of view and no landmark is detected, then the robot continues using the controller's estimates of current pose, starting with the last known transformation it was attempting in order to reach the waypoint. In our case, this means using the PID controller to get the current state, estimate the error to the waypoint, and update the motor commands needed to reach the waypoint.

(f) A short analysis of how smooth your car moves.

We ran into many issues with the pose estimation when the car was mobile that we weren't able to resolve in time for the deadline. Therefore our error was pretty large.

(g) The total moving distance in units you select. For example, if the distance between the first landmark and second landmark is 0.5m, then your unit is 0.5m (The unit should not be less than 0.5m). By the way, the total moving distance (include rotation) is part of your performance.

Total moving distance: 2.474 m.

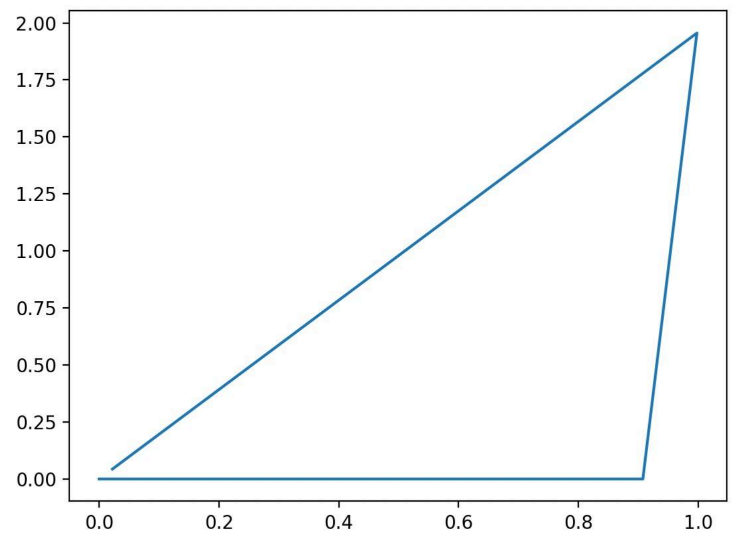
(h) The location error (both rotation and translation) on each waypoint.

Waypoint (x, y, θ)	Localization Error (εx, εy, εθ)

(0, 0, 0)	(0, 0, 0)
(1, 0, 0)	(.0696, -.1830, -.3384)
(1, 2, π)	(.4129, .8799, .8465)
(0, 0, 0)	(.8345, 1.8960 2.9343)

(i) A 2D trajectory of the car. That is, you need to draw a dot into an image for each 0.5 second like the following. (or 1 second based on your speed.) Those positions should be estimated by your algorithm instead of your eyes.

We didn't actually finish so we couldn't create a 2D trajectory. The closest we got is a preliminary estimate of the path, which we did after calibration of the camera. We attached this below:



YouTube Video: Didn't have time to link, but posted our links on Piazza.