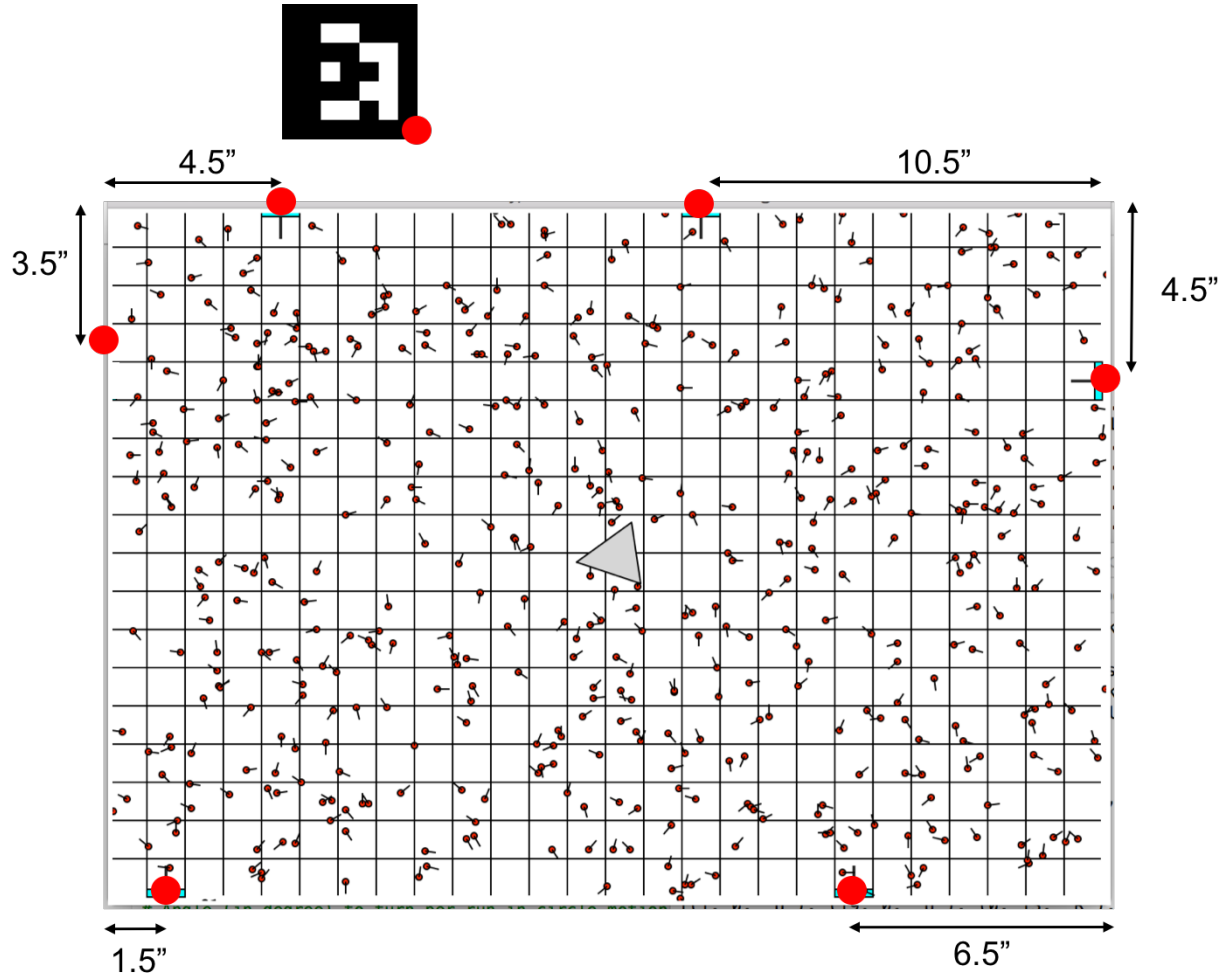
LAB 5: Particle Filter – Part II
Due: Thursday, April 28th (in-class demo)

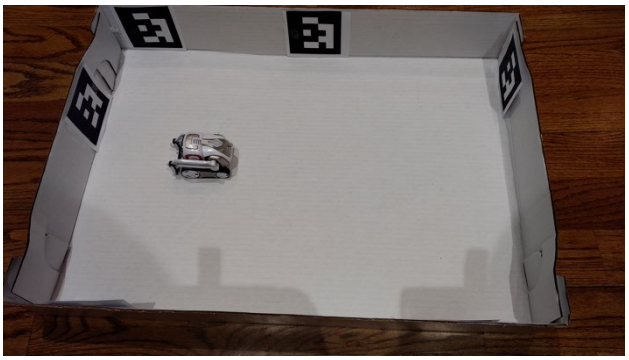
The objective of lab 5 is to implement a **Monte-Carlo Localization** (using Particle Filters). In the second part, you will enable Cozmo to localize within its arena using Particle Filter.

Part II

Setup: Before you begin, you need to add localization markers to the arena. The file `marker.pdf` provides copies of markers we will use. You need six markers, so print out two single-sided copies of `marker.pdf` and cut out six full markers, leaving a small white boarder around the black square. Carefully tape the markers onto your arena such that the corner of the marker highlighted in red below is the appropriate distance from the designated corner.



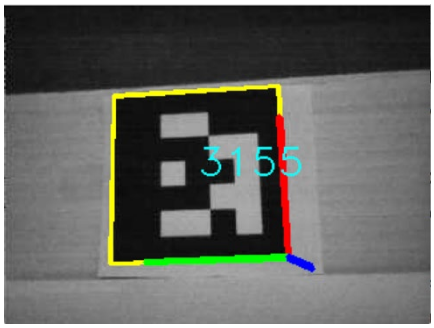
Your completed arena should look like this:



Marker detection: We have provided marker detection code for your use. We do not expect you to edit code in this folder, but if you choose to do so please submit your modified version. To test the code, run

- `python3 test_marker_detection.py`

with Cozmo on. Place Cozmo about 10 inches from a marker, and you should see the marker become highlighted in the image window if it is recognized. Spend a few minutes experimenting with this capability to gain an understanding of the distances and orientations at which the robot is able to detect the marker. Note that the ID displayed for the marker may change, particularly when the robot does not see the marker very well. This is not an issue since the code relies only on the pose of the marker and ignores the ID.



Localization: The main component of the lab is to use the particle filter to enable the robot to 1) determine where it is, and 2) use this information to go to a predefined goal location on the map. We have provided a starter file, called `go_to_goal.py` which contains the following functions:

`image_processing()` : waits for a camera image, runs marker detection on the image

`cvt_2Dmarker_measurements()` : marker processing helper function, calculates the position and orientation of detected markers

`compute_odometry()` : calculates the robot's odometry offset since the last measurement

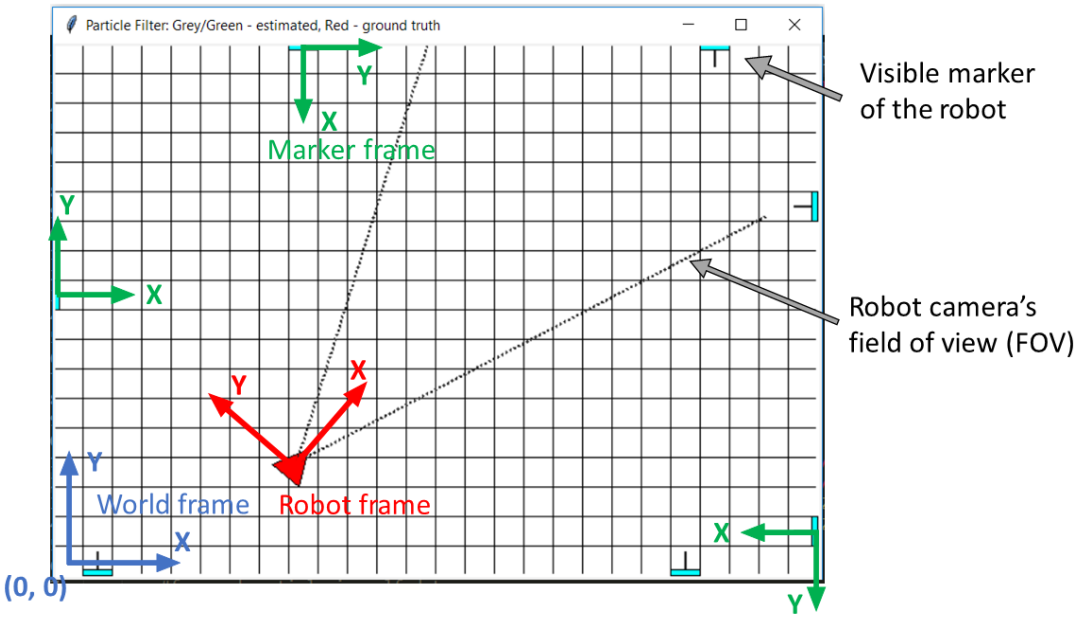
`run()` : the main processing loop, you should enter your code here

The main processing loop should:

- Obtain odometry information
- Obtain list of currently seen markers and their poses
- Update the particle filter using the obtained information
- Update the particle filter GUI for debugging

- Determine the robot's actions based on the current state of the localization system. For example, you may want the robot to actively look around if the localization has not converged (i.e., global localization problem), and drive to the goal if localization has converged (i.e., position tracking problem).
- Have the robot drive to the goal. Note that the goal is defined in terms of both position and orientation. Once there, have the robot play a happy animation, then stand still.
- Make your code robust to the "kidnapped robot problem" by resetting your localization if the robot is picked up. This should trigger both when the robot is on its way to the goal and once it has reached it (i.e., picking up the robot from the goal and placing it somewhere else should immediately cause it to try to localize itself and search for the goal again).

Note that in this part of the lab you may find it useful to leverage some of your earlier implementations of finite state machines and path planning. The coordinate frame that we are using is the same as in prior lab:



Grading: You will demo this part of the lab for grading during class on the day the assignment is due. The assignment will be evaluated for 1) obtaining the correct robot position estimate (location), 2) enabling the robot to successfully drive to the goal. We will grade your submission by looking at the demo and executing your code.

Due to the probabilistic nature of the particle filter, not every run will lead to success. To test the robustness of your code, we will conduct three runs and drop the lowest performing run. In a single execution run, the robot will be placed at start location from which one or more markers are visible and allowed to explore until it reaches the goal or until 90 seconds elapse. At a fixed period of time into the run (e.g., after 20 seconds), your robot will be "kidnapped" and placed in a new location.

If at any point your robot is stuck in a hopeless situation, you may request a "kidnapping" in which case your grader will relocate the robot to a central location in the arena with good view of the markers for a 5-point penalty.

Note that we will not purposefully create problematic situations, such as starting the robot facing a corner.

Extra credit (5 points): robot acts unhappy when being picked up.

Our grading rubric will look like this:

Group #	Run	PF correctly converged	Reached goal	Reset on kidnap	Kidnap request penalty	Extra credit (1-time)	Run Subtotal	Total (sum best 2 of 3)
	1	/30	/15	/5	/-5	/5	/50	/100
	2	/30	/15	/5	/-5	/5	/50	
	3	/30	/15	/5	/-5	/5	/50	

Submission: Submit only your `go_to_goal.py` file with and make sure to enter the names of both partners in a comment at the top of the python file. Only one partner should upload the file to Blackboard. If you relied significantly on any external resources to complete the lab, please reference these in the submission comments. Make sure to include a link to a video of your demo as well.