## Third run of calibration data

Updates:

1. This has a circular obstacle instead of square
2. There is now a track.getObstacleDistance(system\_pos) function that will return your system’s distance to the edge of the circular object
3. Now all the storage variables only include the actual trial. I cut off any extra elements that never got set because the trial ended early (if hit obstacle or went outside track)

BUGS:

1. The car may go the wrong way (clockwise) on a small amount of runs. So remove trajectories that have the **car’s y value < -0.5**. I tried to enforce the car going the right way (counterclockwise) by setting the initial state to have a small upward velocity (0.2), but that didn’t work for all trials

Obstacle position/size:

obstacle = struct('center', [-0.625; 2], 'radius', 0.25);

Obstacle spawn settings:

obstacle\_spawn\_mean = 1; % spawn obstacle once state passes this line

obstacle\_spawn\_var = 0.05;

Obstacle spawn criteria (system\_state(2) is y position):

obj.obstacle\_spawn\_ylim set in constructor as:

mvnrnd(obj.obstacle\_spawn\_mean, obj.obstacle\_spawn\_var, 1);

if(system\_state(2) > obj.obstacle\_spawn\_ylim)

obj.obstacle.active = true;

end

Variable descriptions:

% time array

t\_arr = zeros(1, length(x\_hist(1,:)));

% state history of the actual car

x\_hist = zeros(length(car.x), length(0:1/f\_anc:T\_sim) + 1);

% state history of the MPPI nominal car at each time

x\_mppi\_hist = zeros(length(car.x), length(x\_hist(1,:)));

% the chosen MPPI rollout trajectory (cost-weighted average trajectory)

x\_mppi\_traj\_hist = cell(1, length(x\_hist(1,:)));

% control history of the actual car (MPPI + ancillary + control noise)

u\_tot\_hist = zeros(length(car.u), length(x\_hist(1,:)));

% control history of the MPPI controller

u\_mppi\_hist = zeros(length(car.u), length(x\_hist(1,:)));

% control history of the ancillary controller

u\_anc\_hist = zeros(length(car.u), length(x\_hist(1,:)));

% true if the actual car’s position is outside the track

outside\_track = false(1, length(x\_hist(1,:)));

% true if the obstacle is active

obs\_isactive = false(1, length(x\_hist(1,:)));

% true if the actual car’s position within the obstacle

obs\_hit = false(1, length(x\_hist(1,:)));

## Trajectory plots (plotTrajectory\_final.m):

magenta = hit obstacle

red = goes outside the track

grey = successful

