HOMEWORK 3

JULIEN BLANCHET

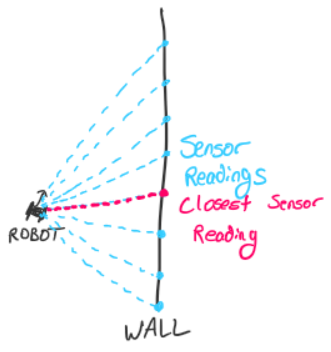
OCTOBER 15, 2019 • COSC 181 • PROF. LI

THE TASK

Assume that you have the Husarion ROSbot 2.0 robot with a 360 degree field of view LIDAR (the measurement is stored in the ROS message LaserScan), modeled it as a differential drive robot. Assume that the robot can be controlled via a ROS Twist message -- the linear velocity is set to a constant 0.5 m/s, while you can control the angular velocity.

QUESTION 1

Consider the wall following problem: the robot must keep following the wall on the right at a given distance (assume that the robot is in an infinitely long corridor, and the robot will not start facing the wall). Given this problem, model for a general closed-loop controller: the state, the actuation command, and the error. Specify how you will use the sensor to measure the state. The answer can be in pseudocode. Draw the robot, the wall, and the sensor readings that are useful to understand your model.



STATE

$$X = \begin{cases} d: \text{distance from wall} & \in \mathbb{R}^+ \\ \theta: \text{orientation} & \in \mathbb{R} \end{cases}$$

ACTUATION COMMAND

$$u = \begin{cases} \theta_z: \text{angular velocity (yaw)} & \in \mathbb{R} \\ v: \text{linear velocity (forward)} & \in \mathbb{R} \end{cases}$$

FINDING DISTANCE TO WALL

```
onBasescanMsgReceive(msg):
    Sensed_points = ComputePoints(msg.angle_increment,
                                  msg.max_angle,
                                  msg.min_angle,
                                  msg.ranges)
    transformed_points = toBaseLinkFrame(sensed_points, msg.header.frame)
    distances = transformed_points.map(p => dist(p, baselink.origin))
    Self.wall_dist = distances.getMin()
```

ERROR

$$e(t) = d_{target} - d_t$$

$\hookrightarrow$ target distance from wall

IMPLEMENTING ACTUATION COMMAND

```
def move(ang-vel, lin-vel):
    L = 0.235   # given in meters
    V_left = lin-vel - L/2 * ang-vel
    v_right = lin-vel + L/2 * ang-vel
```

$\omega = \frac{v_r - v_l}{l} = \text{ang\_vel}$

$v_{net} = \frac{v_r + v_l}{2} = \text{lin\_vel}$

$\hookrightarrow v_l = 2 \cdot \text{lin\_vel} - v_r$

$v_r - v_l = l \cdot \text{ang\_vel}$

$v_r - (2 \cdot \text{lin\_vel} - v_r) = l \cdot \text{ang\_vel}$

$2v_r = l \cdot \text{ang\_vel} + 2 \cdot \text{lin\_vel}$

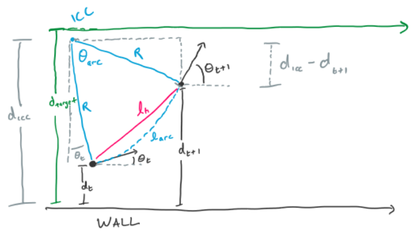
$\boxed{v_r = \text{lin\_vel} + \frac{l}{2}\text{ang\_vel}}$

$v_l = 2 \cdot \text{lin\_vel} - (\frac{l}{2} \cdot \text{ang\_vel} + \text{lin\_vel})$

$\boxed{v_l = \text{lin\_vel} - \frac{l}{2}\text{ang\_vel}}$

QUESTION 2

A discrete P controller is used to control the robot with a proportional gain of 1. The robot starting pose is exactly parallel to the wall and the laser sensor returns a measurement of 1.5m to the right of the robot. How long will it take to get the robot to the target distance of 2m from the wall, using the model you defined in 1.? Show your calculations and the error over time, at a sampling rate of 0.2 seconds. You can assume that the robot follows the forward kinematics model for the differential drive robot.

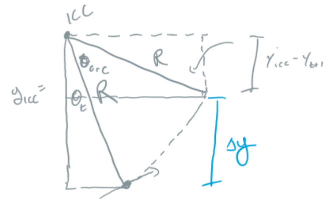


THE CONTROLLER

$$u \begin{cases} \theta_z = 2m - d \\ v = 0.5 \text{ m/s} \quad \text{(given)} \end{cases}$$

The Transition Function

$f(\theta_t, d_t, w, v, \Delta t) \rightarrow \theta_{t+1}, d_{t+1}$



$y_{icc} = d_t + R \cdot \cos(\theta_t)$

$R \cdot \cos(\theta_{arc} + \theta_t) = d_{icc} - d_{t+1}$

$d_{t+1} = d_{icc} - R \cdot \cos(\theta_{arc} + \theta_t)$

$\omega = d_{target} - d_t$

$v = 0.5 \text{ m/s}$

$\Delta t = 0.2s$

$l_{arc} = v \cdot \Delta t$

$\theta_{arc} = \omega \cdot \Delta t = \frac{\theta_{arc}}{R}$

$\boxed{\theta_{t+1} = \theta_t + \theta_{arc}}$

$R = \frac{l_{arc}}{\theta_{arc}}$

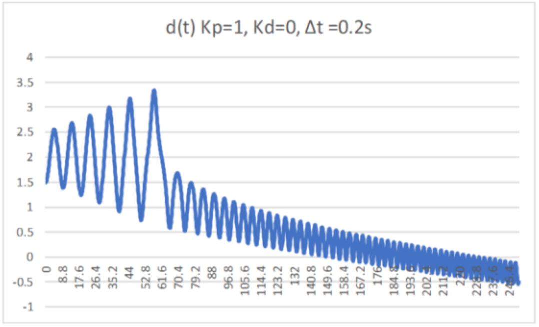
$d_{icc} = d_t + R \cdot \cos(\theta_t)$

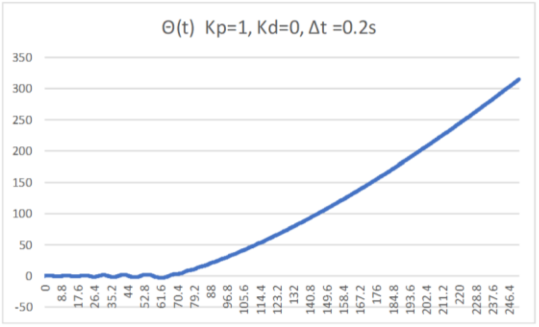
$d_{icc} - d_{t+1} = R \cdot \cos(\theta_{arc} + \theta_t)$

$\boxed{d_{t+1} = d_{icc} - R \cdot \cos(\theta_{arc} + \theta_t)}$

Simulation Results ($K_p = 1$, $K_i = 0$, $K_d = 0$, $\Delta t = 0.2s$)

Time	Θ(t)	d(t)	e(t)	w(t)	Θ(t+1)	d(t+1)	Θ(arc)	R	d(ICC)	deriv(t)	integ(t)
0	0	1.5	0.5	0.5	0.1	1.504996	0.1	1	2.5	0	0.5
0.2	0.1	1.504996	0.495004	0.495004	0.199001	1.519884	0.099001	1.010093	2.510042	-0.005	0.995004
0.4	0.199001	1.519884	0.480116	0.480116	0.295024	1.544326	0.096023	1.041415	2.540747	-0.01489	1.47512
0.6	0.295024	1.544326	0.455674	0.455674	0.386159	1.577718	0.091135	1.097275	2.594193	-0.02444	1.930794
0.8	0.386159	1.577718	0.422282	0.422282	0.470615	1.619247	0.084456	1.184044	2.674573	-0.03339	2.353076
1	0.470615	1.619247	0.380753	0.380753	0.546766	1.667938	0.076151	1.313186	2.789675	-0.04153	2.733829
1.2	0.546766	1.667938	0.332062	0.332062	0.613178	1.722728	0.066412	1.505744	2.95416	-0.04869	3.065891
1.4	0.613178	1.722728	0.277272	0.277272	0.668633	1.782513	0.055454	1.803285	3.197496	-0.05479	3.343163
1.6	0.668633	1.782513	0.217487	0.217487	0.71213	1.846191	0.043497	2.298984	3.586457	-0.05978	3.56065
1.8	0.71213	1.846191	0.153809	0.153809	0.742892	1.912689	0.030762	3.25078	4.306939	-0.06368	3.71446
2	0.742892	1.912689	0.087311	0.087311	0.760354	1.980971	0.017462	5.726683	6.13048	-0.0665	3.80177
2.2	0.760354	1.980971	0.019029	0.019029	0.76416	2.050027	0.003806	26.27581	21.02022	-0.06828	3.820799
2.4	0.76416	**2.050027**	-0.05003	-0.05003	0.754154	2.118857	-0.01001	-9.99468	-5.16577	-0.06906	3.770772

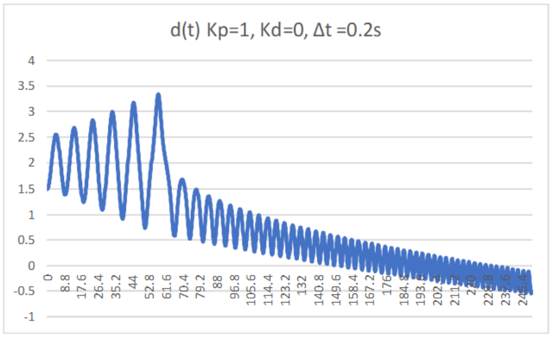


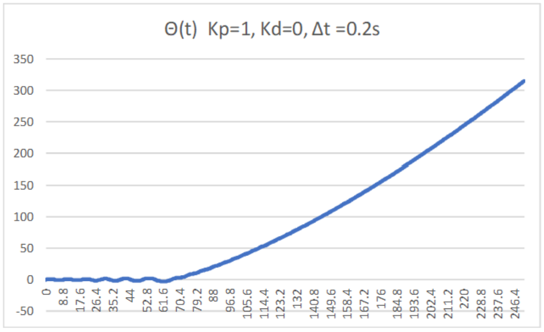


QUESTION 3

Considering the scenario in 2., is the robot overshooting? Please motivate the answer, and if you believe it will overshoot, show an option to avoid overshooting, simulating again the scenario in 2.

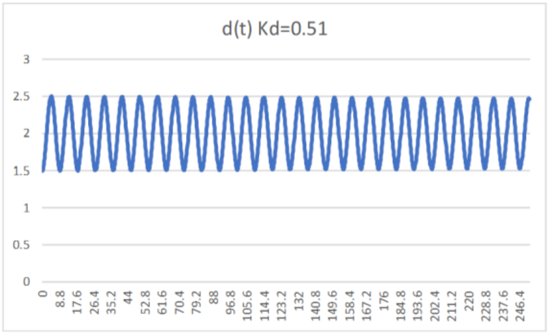
The robot is overshooting. This can be easily observed if one simulates the model for a longer period of time, as shown in the graph below (simulated with Kp=1 for 245 seconds). After 4.8s, the error is -0.55m, which is greater in amplitude than the initial error of 0.5m. From the chart, one can observe that the amplitude of the error fluctuation increases over time, up to 66s, at which point the error has reached ~1.42m and the robot is sufficiently far from the target distance to complete full circles before returning the target distance.

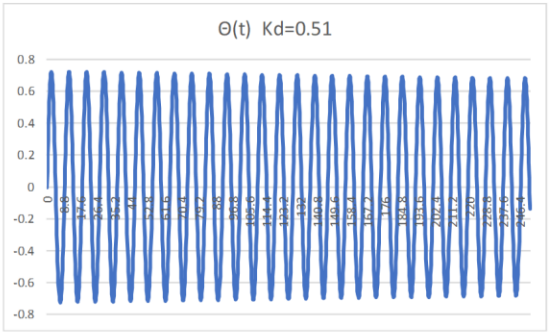


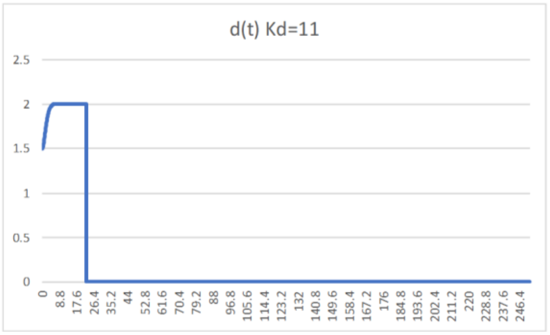


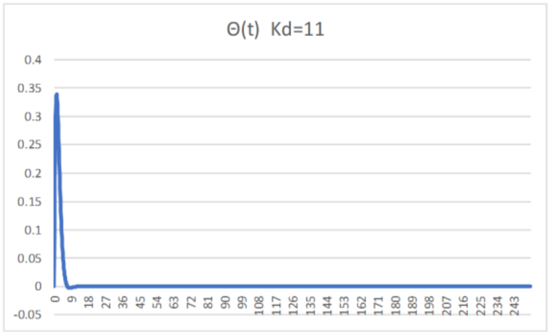
The most straightforward way to address this is to incorporate a derivative term. I experimented with a few derivative terms, and found that a term of Kd=0.51 was the minimum sufficient to lead to eventual stabilization (so that the error amplitude decreases), and that Kd=11 was sufficient to prevent overshooting. (see charts below).

- Note that all the below charts have Kp set to 1
- Note that the sudden drop at t=22.6s for the Kd=11 simulation is due to the limitations of excel floating point precision: the error is small enough for the angular velocity / ICC / R / arclength calculations to fail with a division by zero error.









QUESTION 4

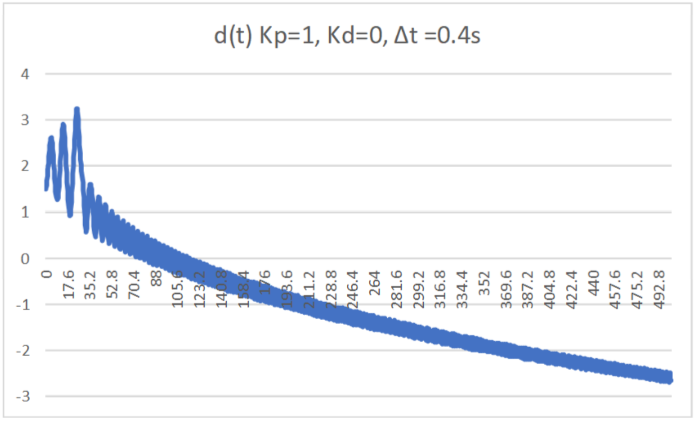
Considering again the scenario in 2., what happens if the sampling rate becomes 0.4 seconds? Repeat the calculations and show the error over time.

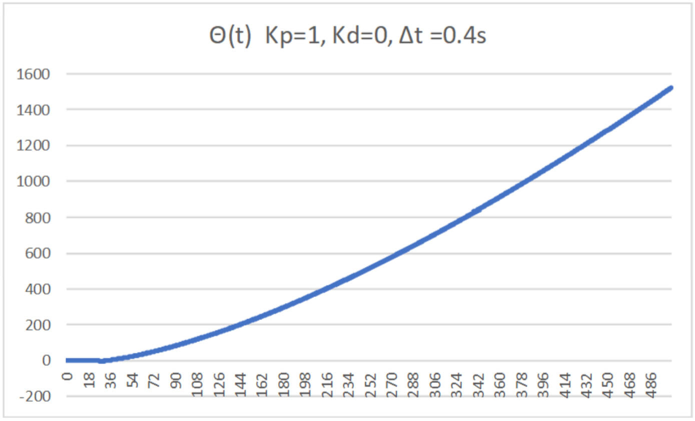
Looking at the calculations below, one can observe that the target goal is still passed by t=2.4s, and by a slightly higher amount. My hypothesis for why this is the case is that the reduced sampling rate causes the robot to go longer with a higher angular velocity before "recomputing" and slowing down the turn rate as it approaches the target distance.

Looking further along the simulation, one can also observe that the simulation breaks at an earlier point (around t=32.8s, compared to t=66s with the 0.2s sampling rate).

Simulation Results $K_p=1, K_i=0, K_d=0, \Delta t = 0.4s)$

Time	Θ(t)	d(t)	e(t)	w(t)	Θ(t+1)	d(t+1)	Θ(arc)	R	d(ICC)	deriv(t)	integ(t)
0	0	1.5	0.5	0.5	0.2	1.519933	0.2	1	2.5	0	0.5
0.4	0.2	1.519933	0.480067	0.480067	0.392027	1.578186	0.192027	1.041522	2.540695	-0.01993	0.980067
0.8	0.392027	1.578186	0.421814	0.421814	0.560752	1.669792	0.168726	1.185356	2.673616	-0.05825	1.401881
1.2	0.560752	1.669792	0.330208	0.330208	0.692836	1.787017	0.132083	1.514196	2.952096	-0.09161	1.732089
1.6	0.692836	1.787017	0.212983	0.212983	0.778029	1.921158	0.085193	2.347601	3.59335	-0.11722	1.945073
2	0.778029	1.921158	0.078842	0.078842	0.809566	2.063756	0.031537	6.341768	6.438388	-0.13414	2.023915
2.4	0.809566	**2.063756**	-0.06376	-0.06376	0.784064	2.206779	-0.0255	-7.84241	-3.34604	-0.1426	1.960159
2.8	0.784064	2.206779	-0.20678	-0.20678	0.701352	2.341997	-0.08271	-2.41804	0.494683	-0.14302	1.75338
3.2	0.701352	2.341997	-0.342	-0.342	0.564553	2.460211	-0.1368	-1.462	1.225072	-0.13522	1.411383
3.6	0.564553	2.460211	-0.46021	-0.46021	0.380469	2.551107	-0.18408	-1.08646	1.54234	-0.11821	0.951173





QUESTION 5

In the wall following problem is there any reason to introduce the integral term? Please motivate your answer.

I don't believe that an integral term would help would this problem, largely due to the simplicity of the simulation. We saw in class that in the case of a PID controller for a drone seeking to achieve a certain altitude, an integral term was useful to account for the constant RPM that would be necessary to sustain the target altitude. In this problem, there is no such external force (i.e. gravity) that would introduce issues with set point tracking; when the robot reaches the target distance from the wall and is parallel to it, no further turning or actions are required beyond simply proceeding directly forward.[1]

Additionally, I tried simulating many Ki factors and found that the smaller factor I had, the faster the robot converged on the target distance. A Ki of zero seems to be ideal in this case.

[1] I consulted this web article while writing my answer: https://controlstation.com/using-the-integral-term/