Computer Science 685 Midterm

Paul McKerley (G00616949)

November 12, 2018

1. If I understand the question correctly, it is to define the forward kinematics of the point at the end of $L2$. Since this is the position of the gripper, I will refer to it as G. I interpret the diagram to mean that $L1$ is a fixed bar, which slides up and down $y1$, causing $d1$ to vary. Also, θ_2 can change, causing $L2$, which is also fixed, to swing around.

   So the position of G in the second coordinate frame is simply how far it is translated from the end of $L1$, which is

   $$G_2 = \begin{bmatrix} L2 \\ 0 \end{bmatrix}$$

   Now in the first coordinate frame, G is rotated by θ_2 and translated by $L1$ on the x axis and $d1$ on the y axis. So

   $$G_1 = \begin{bmatrix} L1 \\ d1 \end{bmatrix} + R(\theta_2) \cdot \begin{bmatrix} L2 \\ 0 \end{bmatrix} = \begin{bmatrix} L1 \\ d1 \end{bmatrix} + \begin{bmatrix} L2cos(\theta_2) \\ L2sin(\theta_2) \end{bmatrix} = \begin{bmatrix} L1 + L2cos(\theta_2) \\ d1 + L2sin(\theta_2) \end{bmatrix}$$

   As mentioned above, the kinematic parameters are $L1$, $L2$, and implicitly the angles of $d1$ with respect to $x1$, and $L1$ with respect to to $d1$. The joint variables are $d1$ and θ_2.

2. a) The function needs U_r to be expanded to include repulsion from robot $R2$ in position x_2. So it would look something like
   $U(x_1) = k_1||x_1 - x_g||^2 + k_2 \frac{1}{||x_1 - x_o||^2} + k_3 \frac{1}{||x_2 - x_1||^2}$
   with the repulsion of the $R2$ possibly having a different constant k_3, since a moving obstacle might require speedier avoidance.

   b) The control function will be to figure out the forces applied to $R1$ at that position. We need to define distances of influence for the obstacle and the other robot, so, we will define q_o and q_2 to be the minimum distances from the obstacle and other robot, respectively, before they have a repulsive effect.
   $F(x_1) = -k_1 \cdot (x_1 - x_g)$

   $$- \begin{cases} k_2(\frac{1}{||x_1 - x_o||} - \frac{1}{q_o}) \frac{1}{||x_1 - x_o||^2} \frac{x_1 - x_o}{||x_1 - x_o||}, & \text{if} ||x_l - x_o|| \leq q_o \\ 0, & \text{otherwise} \end{cases}$$

   $$- \begin{cases} k_3(\frac{1}{||x_1 - x_2||} - \frac{1}{q_o}) \frac{1}{||x_1 - x_2||^2} \frac{x_1 - x_2}{||x_1 - x_2||}, & \text{if} ||x_l - x_2|| \leq q_1 \\ 0, & \text{otherwise} \end{cases}$$

   c) The repulsive and attractive forces at a certain location can cancel each other out perfectly, so that the net force applied at that location is 0; ie, the robot encounters a local mininum. When this happens the robot could adopt a number of strategies. When if finds itself stuck, it could go on a random walk and hope to leave the local minimum. It could use a strategy like one of the Bug algorithms. For example, the Bug2 algorithm tells the robot that when it can't move, it should remember its bearing to the goal, and circle around the obstacle until it hits that bearing line again, at which point it should continue toward the goal.

3. I'm assuming that the d_x^2 and d_z^2 variables are constant for a given run. It seems that the variance of our state converges quickly to a stable value. As d_x^2 and d_z^2 approach zero, the point of stabilization becomes closer and closer to zero. I interpret this to mean that as the variances in the perturbation and sensor readings become more and more accurate (ie, d_x^2 and d_z^2 approach zero), then the robots estimate of its position likewise becomes more accurate, and the variance likewise approaches zero.

   Here is the code I used to create the graphs:

```
#!/usr/bin/env python
import matplotlib.pyplot as plt
import pdb
import sys

def filter_t_plus_1(dt2, dx2, dz2):
    return ((dt2 + dx2) * dz2)/(dt2 + dx2 + dz2)

def main(dx2, dz2, dt2_init):
    ts  = [1.0]
    dt2 = [dt2_init]
    for t in range(1,1000):
        dt2.append(filter_t_plus_1(dt2[-1], dx2, dz2))
        ts.append(t)

    plt.plot(ts, dt2)
    plt.title("dx2={0}, dz2={1}, starting value of dt2={2}".format(dx2, dz2, dt2_init))
    plt.savefig('q3_dx2_{0}_dz2_{1}_dt2_init{2}.pdf'.format(dx2, dz2, dt2_init), dpi=600)

if __name__=='__main__':
    main(float(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3]))
```

4. a) $x_2^T \hat{T} x_1 = 0$, where $\hat{T} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$.

   b)

   c)

5. I wasn't exactly sure how to do this, so I based it on the diagram on p 34 of the cs685-sensor-models Powerpoint. The only hitch is that the cell towers aren't on the same X position, but I think I got the geometry right to compute where the cell tower measurements intersect. I wrote a Python script to calculate it, because I didn't want to make mistakes, and I used scipy.stats to find the probabilities of the friend being a certain distance from each cell tower (with the combined variance from the two distributions.) I've included the code below. I'll also attach the graph paper I used to sketch out the problem.

   - With no probability information, simply the distance that the intersection of the cell signal intersections is from each location is enough to determine which is more likely. I calcalated that the Johnson Center is 1.675 distant, and home is 1.497 distant. So she is more likely to be at the Johnson Center.
   - I combined the variances from the two distributions, for the points of overlap, and got 0.6. I then used the normal PDF for the values above, and multiplied them be the probabilities for being at home or school, 0.7 and 0.3, respectively, normalized, and got the likelihood that shes at home as 0.52 and at the Johnson Centeras as 0.48.

```
#!/usr/bin/env python
import math
import pdb
import scipy.stats as stats

def dist(p1,p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def main():
    x0 = (12, 4)
    x1 = (5, 7)
    p0 = (10, 8)
    p1 = (6, 3)
    d0 = 3.9
    d1 = 4.5

    # Compute intersection points analagously to p 34 of
```

```
    # cs685-sensor-models.ppt. But have to account for rotation.

    # distance between cell towers
    a = dist(x1, x0)

    x = (a**2 + d0**2 - d1**2)/(2*a)

    # Use similar triangles to get p2
    p2 = (x1[0] + (x0[0] - x1[0]) * x/a,
          x0[1] + (x1[1] - x0[1]) * x/a)

    # angle between cell towers
    theta = math.atan2(x1[1] - x0[1], x0[0] - x1[0])

    # calculate y value from ppt
    y = math.sqrt(d0**2 - x**2)

    x_delta = y * math.sin(theta)
    y_delta = y * math.cos(theta)

    # Find intersection points
    int0 = (p2[0] - x_delta, p2[1] - y_delta)
    int1 = (p2[0] + x_delta, p2[1] + y_delta)

    # Find closest home distance
    home_dist = min(dist(p1,int0), dist(p1, int1))

    # Find closest Johnson distance
    jc_dist   = min(dist(p0,int0), dist(p0, int1))

    print("Home is {} distance from intersection of cell signals".format(home_dist))
    print("Johnson Center is {} distance from intersection of cell signals".format(jc_dist))
    if jc_dist < home_dist:
        print("Johnson Center is most likely before prior knowledge.")
    else:
        print("Home is most likely before prior knowledge")

    # Combine variances using formula 5.73 on p 330 of Siegwart
    var0 = 1.0
    var1 = 1.5
    var2 = var0 - var0**2/(var0 + var1)
    print("combined variance is {}".format(var2))

    norm = stats.norm(0, var2)
    p_jc   = norm.pdf(jc_dist) * 0.3
    p_home = norm.pdf(home_dist) * 0.7

    p_total = p_jc + p_home
    p_jc = p_jc/p_total
    p_home = p_home/p_total

    print("With prior knowledge, probability of being at Johnson Center is {}".format(p_jc))
    print("With prior knowledge, probability of being at Home is {}".format(p_home))

    return

if __name__=='__main__':
    main()
```

6. a) $2r = 16mm$, so $r = 8mm$. $f = 16mm$. So the FOV $\theta = 2\arctan(r/f) = 2\arctan 2(8, 16) \approx 0.927295$.

   b) With $k_u = \frac{640px}{0.016m} = 40000px/m$, $k_v = \frac{480px}{0.012m} = 40000px/m$, $u_0 = 320px$, $v_0 = 240px$, $f = 0.016m$,

   $$K = \begin{bmatrix} fk_u & 0 & u_0 & 0 \\ 0 & fk_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.016 \cdot 400000 & 0 & 320 & 0 \\ 0 & 0.016 \cdot 40000 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 640 & 0 & 320 & 0 \\ 0 & 640 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

   To calculate a point (u, v) in the image coordinate system from a point in the world coordinate system (x, y, z) we would perform the following calculation:

   $$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} 640 & 0 & 320 & 0 \\ 0 & 640 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

   Where we get the final answer $\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda^{-1} \cdot \begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix}$.

   Of course this ignores consideration of the rigid body transformation between the world coordinate system and the camera coordinate system (because the question has no information about that.)

7. - There can be three independent rotations around, and three independent translations on, each of the coordinate axes, which gives a total degrees of freedom of 6.
   - Given two views of the same scene, two points are in correspondance if the represent the same point in the real world. Some difficulties establishing correspondance include
     - The relative positions of the cameras taking the images are unknown. This might mean the the scene point is subject to significantly differnt RBTs between the two images.
     - The point may be difficult to distinguish from other similar points in the scene (for instance, the point may be on a large expanse of uniform color.)
     - The conditions under which the two images were captured may be different–for example, the lighting may have changed.
     - The images may have been captured with cameras having different intrinsic properties.
   - I think there would be an aspect of Gaussian uncertainty in a typical measurement of an object. But I don't think the uncertainty itself could be represented by a Gaussian distribution. This is because this is only one of many factors causing errors in the measurements, such as:
     - The maximum range of the sensor means any attempt to measure beyond a certain distance would return the same value.
     - Transitory occlusions, like people moving in front of the camera.
     - Interference from other ultrasound sensors would cause non-Gaussian errors.

8. The equation determines how to control a system based on error feedback. The term $k_p e$ is applying control proportional to the error. $k_d \dot{e}$ applies control proportional to the rate of change of the error (derivative); $k_i \int e(t)dt$ accumulates error over time and applies control based on that (integral). The control coefficients k_p, k_d, and k_i are gains, and emphasize each control factor according to how we want the system to behave.

   Take for instance the in-class example of a PID controller for a car's cruise control. Our goal is to attain a certain speed. Then e is how far we are from the speed we want. One thing we want to do is press the accelerator when we are far from the speed, which is what the proportional controler is for. The further we are from our from our goal speed the faster we accelerate. This may lead to jerky acceleration and over-shooting, so we can use the derivative control to modulate the acceleration when the rate of change is too fast (we're accelerating too much.) The integral control accumulates error over time and can be used to compensate for constant error, such as tire friction on the road or wind resistance. The particular gains would have to be set experimentally with the working system to get a desired effect (like a sense of smoothness in acceleration or deceleration.)

9. For a Markov process the robot's probability table measuring the likelihood of it being in each possible state would be completely wrong after kidnapping. But as it moved through its environment, it would start to readjust it's state probabilities to accurately reflect reality. If it were setting states to zero for efficiency, and it were placed in one of those states, it might be impossible, or nearly so, for the robot to get those states back into to non-zero, regardless of its sensor inputs. If the kidnapping occured before the robot had performed much sensing, and its state probabilities were close to the initial uniform distribution, then it would be more likely to recover. In general however, it would be able to recover.

   Kalman processes must always know their initial positions and cannot recover if lost, so it would be unable to recover if kidnapped and placed in a new location.