Intro to Mobile Robots - Spring 2017

Homework #1

Note to myself - This file has conditional text formatting to hide the answers.

Generally, I'd prefer if you submit all homeworks entirely in electronic form so I can grade them wherever I am. PDF files are best. See the web page for how to get PDF tools from Adobe for free.

This homework is really just a set of exercises to generate intuition for the properties of homogeneous transforms. The answers to every question occur somewhere in the course notes or text. You are well advised to work through each one, alone, in great detail until you get the point. Future homeworks will assume you know this material very well.

You should do this and all homeworks alone without any undue help from your fellow students. Help with logistics like setting up Eclipse is OK.

Grades in this course are based mostly on homeworks. Therefore, don't make the mistake of ignoring any questions asked. Answer each one explicitly.

Homogeneous Transforms

2D Homogeneous transforms work just like 3D ones except that a rigid body in 2D has only three degrees of freedom - translation along x or y or rotation in the plane. Consider the transform:

$$T = \begin{bmatrix} 0 & -1 & 7 \\ 1 & 0 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

This will be "the transform" for the rest of this homework.

In this homework, you will implement elements of the kinematics of 2D homogeneous transforms while studying operator-transform duality and the isomorphism of poses and transforms.

Submit your results in files named {OT_2D,Pt_2D,Pose_2D}.YourAndrewId.java etc., and Hw1.YourAndrewId.pdf (or doc or whatever). You can also submit paper for the external documentation if you cannot get PDF to work.

Read and work through all of the steps in the file GettingStartedInJava.doc to set up your environment. Once you send an email for a license, you may have to wait several days for an answer, so work the paper exercises before you try to work the coding exercises. You will not be able to write the code necessary until you have mastered the concepts in the paper exercises (exercises 1-4). Your hand drawn graphs should probably be sketched quickly at first because you can use the screen captures from the programming exercise to replace them if your code works.

1. Operators and Frames [20 points]

1.1 Operating on a Frame [5 points]

Recall that the unit vectors and origin of a frame can be represented in its own coordinates as an

identity matrix. Let such a matrix represent frame ‘a’. Consider “the transform” matrix to be an operator and operate on the unit vectors and origin of frame ‘a’ expressed in its own coordinates to produce another frame, called ‘b’.

a) Write explicit vectors down for the unit vectors and origin of the new frame ‘b’. Use the notation which records the coordinate system in which they are expressed.

1.2 Visualization of the New Frame [5 points]

When a transform is interpreted as an operator, the output vector is expressed in the coordinates of the original frame.

a) Get out some graph paper or draw a grid in your editor with at least 10 X 10 cells. Draw a set of axes to the bottom left of the paper called frame ‘a’. Draw the transformed frame, called ‘b’ in the right place with respect to frame a based on the above result. Label the axes of both frames with x or y.

1.3 Homogeneous Transforms as Frames [10 points]

Consider the coordinates of the unit vectors and origin of the transformed frame when expressed with respect to the original frame.

a) Compare these coordinates to the columns of the homogeneous transform. How are they related?

b) Explain why this means homogeneous transforms are also machines to convert coordinates of general points under the same relationship between the two frames involved. HINT: how is a general point related to unit vectors and origin of any frame.

2. Pose of a Transform and Operating on a Point [25 points]

2.1 Solving for the Relative Pose [10 points]

The parameters of the compound homogeneous transform which relates these two frames can be found using the techniques of inverse kinematics.

a) Write an expression (in the form of a homogeneous transform with three degrees of freedom (or “parameters” in operator form) $\begin{bmatrix} a & b & \theta \end{bmatrix}^T$ for the general relationship between two rigid bodies in 2D, equate it to the above transform.

b) Solve it by inspection for the “parameters”.

2.2 Rigid Transforms[15 points]

Operating on a general point is no different than operating on the origin.

a) Operate on the point $\underline{p} = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}^T$ with the transform and write the coordinates $\underline{p}'$ of the new point. Copy your last figure and draw $\underline{p}$ and $\underline{p}'$ on it. Label each.

b) How do the coordinates of $\underline{p}'$ in the new frame (called b) compare to the coordinates of $\underline{p}$ in the old frame (called a)?

c) What property of any two points is preserved when they are operated upon by an orthogonal

transform and what does this imply about any set of points?

3. Homogeneous Transforms as Transforms [40 points]

3.1 Transforming a General Point [40 points]

This question is worth 40 points. Make sure you get it right.

a)[10] Copy the last figure including points p and p' on a fresh sheet. Draw the point $\underline{q}^b = \begin{bmatrix} 4 & 1 & 1 \end{bmatrix}^T$. The notation superscript-b means the point has been specified with respect to frame b, so make sure to draw it in its correct position with respect to frame b.

b) [10] Write out the multiplication of this point by "the transform" and call the result $\underline{q}^a$. Using a different symbol on the graph than the one drawn for $\underline{q}^b$, draw $\underline{q}^a$ in its correct position with respect to frame a.

c) [10] Earlier, when p was moved to p', p was expressed in frame a and so was p'. Here you expressed $\underline{q}^b$ in frame b to produce a result $\underline{q}^a$ expressed in frame a. Discuss how *interpreting the input differently (i.e in different coordinates)* leads to a different interpretation of the *function of the matrix*.

d) [10] How can the function performed on the point $\underline{p}$ be reinterpreted as a different function applied, instead, to p'?

4. Inverse of a Homogeneous Transform [25 points]

4.1 Inverse Identity [5 points]

Recall that matrices can be multiplied in block partitioned form - provided the partitions have conformable dimensions. For example, the matrix equation:

$$A = BC$$

Can be written in block partitioned form as:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} (B_{11}C_{11} + B_{12}C_{21}) & (B_{11}C_{12} + B_{12}C_{22}) \\ (B_{21}C_{11} + B_{22}C_{21}) & (B_{21}C_{12} + B_{22}C_{22}) \end{bmatrix}$$

provided all of the component multiplications have appropriate dimensions for multiplication to be defined.

a) Verify the formula for the inverse of a general homogeneous transform given in class. That is, show that it is correct in the general case. "Verify" does not mean "prove". There is no need to *derive* the formula, just show that it always works.

4.2 Transform Inversion [5 points]

Using the general formula given in class, invert "the transform" and write down the value of the inverse matrix.

a) Draw the figure of exercise 1.2 with frame b moved and rotated into standard position (bottom of page, x-right, y-up) and frame a in the same position *relative to it* as the position it has in exercise 1.2. Interpret the columns of the inverse matrix that you wrote.

4.3 Pose Inversion [15 points]

Recall that a pose and a Homogenous Transform are two views of the same fundamental idea. Just as a homogeneous transform has an inverse, so does a pose. One way to understand what an inverse pose means is to write:

$$\underline{\rho}^{-1} = \underline{\rho}\{T\{\underline{\rho}\}^{-1}\}$$

where $\underline{\rho}\{T\}$ means the pose encoded in T and $T\{\underline{\rho}\}$ means the homogeneous transform that corresponds to a given pose. Hence the inverse of a pose is the pose of ... the inverse of ... the transform of ... the original pose.

a) [10] Write a general formula for the inverse $\underline{\rho}^{-1}$ of the pose $\underline{\rho} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ by writing the formula for the homogeneous transform which corresponds to $\underline{\rho}$, then inverting it symbolically using the formula for the inverse of a homogeneous transform, and then reading the parameters of the inverse pose from the inverse matrix. You are looking for three simple formulas that express the parameters of the inverse of a pose in terms of the parameters of the pose itself. HINT: $\sin(\theta) = -\sin(-\theta)$

b) [5] Check the pose inversion formula against your result for "the Transform" in Question 4.2 a). Show your work. Substitute the pose of b wrt a into the formula.

5. Implementing Transform Classes in Java [55 points]

The file applicationHW6Transforms.java is provided in source form. It contains the main driver routines including world creation, graphics etc. It configures a GUI and a drawing canvas using the Simple Mobile Robot Development Environment ("smrde") to produce the figures needed in the above exercises using Java code. You do not need to understand the code very well. Your job will be to fill in a few small holes in the code based *on what you learned above* in order to produce the results of the above exercises one more time.

If you have not done the above exercises, go back and do them first.

For those who care to understand the graphics in the main program, it won't make sense until you note that Smrde uses recursive specification of geometry to construct objects. Thus, a Sprite may contain other Sprites, which themselves may contain others. A hierarchy is created where objects in each level have their poses specified with respect to their immediate parents in the hierarchy. In this way, you specify where the robot is in the world and where the wheels are on the robot, and the software keeps track of where the wheels are in the world once you move the robot. This technique is used in a few places. The key is to check what object's `add()` method is being invoked to see who is stored inside of whom. See the Smrde javadocs (section 2.2) for more. You can read the javadocs in Eclipse by double clicking on the doc>overview.html to have them come up in the built-in HTML browser.

Javadocs are provided for the entire system. It is almost impossible to proceed without figuring out how to access the javadocs. The best way is to configure Eclipse so that it integrates them with the source code editor as outlined in GettingStartedInJava.doc on the course web page. Doing this will save you hours and allow you to see the javadocs on anything including the run-time library by hovering the mouse over the variable of interest. Understanding all of the SMRDE environment will take more time than you have, so don't spend too much time with that. Read what you need when you need it.

Figure 1 shows the main files of interest for this assignment:

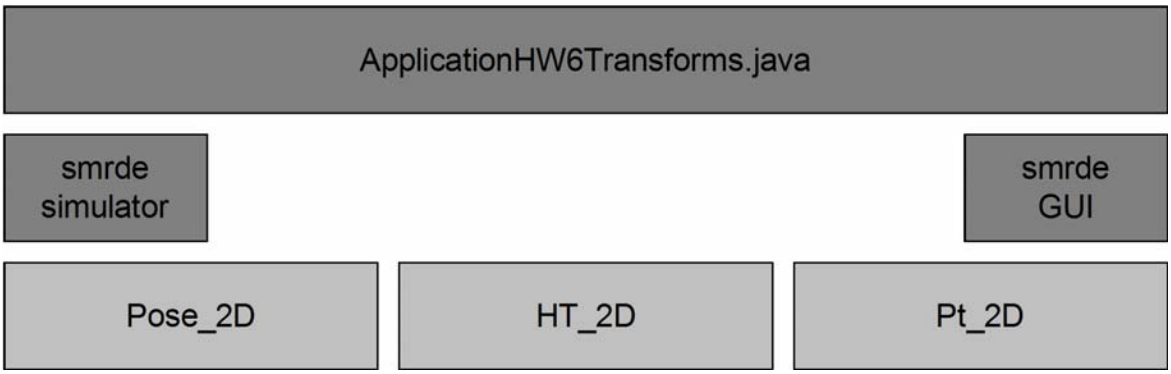

Figure 1: Rough Invocation Hierarchy for the Software for this Assignment.

- `applications.homeworks.ApplicationHW6Transforms` - the main program tied to the Orthog. Transforms application button. Most of the code that is related to setting up the graphics and simulation is in here. You should need to do nothing in here.
- `applications.homeworks.unlocked.hw6.OT_2D.java`. An implementation of a 2D homogenous transform class. It has a few things deliberately missing for you to fill in.
- `applications.homeworks.unlocked.hw6.Pose_2D.java`. An implementation of a 2D pose class. It also has a few things deliberately missing for you to fill in.
- `applications.homeworks.unlocked.hw6.Pt_2D.java`. An implementation of a 2D (homogeneous) point class.

In java's `Math` package, look at `sqrt(), cos(), sin()`. The most straightforward way to print information is with something like:

```
System.out.printf("The variable is: %f\n",myDoubleVariable);
```

Run the program and then press the Homeworks button and then the Orthog. Transforms Button. You should see a small application control GUI and a graphing area in a drawing canvas. Read the instructions in the instructions text area. Note the capacity to capture the drawing canvas to a jpg file. If you get your code to work, you can use these images instead of your hand-drawn graphs produced in questions 1-4. The point of having you draw them by hand was to create an expectation for the right answer. You can't debug a program if you don't know the right output.

If you sequence through the four steps you will notice that they run but that the graphs are not right etc. Your job is to fill in the empty parts of several functions in order to make the program work.

5.1 Part 1 [10 points]

Edit the file OT_2D.java to complete the following functions ...

```
concat()
```

```
XUnitVector()
YUnitVector()
OUnitVector()
```

... and produce the graph for question 1. You can screen capture it (if you believe it is correct) and include it as your drawing for question 1. Do this screen capturing for the following questions as well.

5.2 Part 2 [10 points]

Edit the file OT_2D.java to complete the following functions ...

```
operate()
```

```
x()
y()
th()
```

... and produce the graph for question 2. Look in Pt2D for hints on how this is done for a point.

5.3 Part 3 [10 points]

Edit the file OT_2D.java to complete the following functions ...

```
transform()
```

... and produce the graph for question 3. This is really easy.

5.4 Part 4 [25 points]

This part should already produce the graph for part 4 if you generated the graph for part 1 correctly. However, there are some printed outputs to be generated as well.

Edit the file OT_2D.java to complete the following functions ...

```
inverse()
```

Notice that the file Pose_2D.java has not been used yet. Edit the file Pose_2D.java to complete the following functions...

`inverse()`

Use your result from question 4.3.

You will see that the code in applicationHW6Transforms.java for part4 multiplies the homogeneous transform for the frame at `(1.0,2.0,Math.toRadians(30.0))` by its own inverse to print out the result. It also does a similar thing with the equivalent pose.

Provide a transcript of your Eclipse Console window output and argue its correctness by answering the following questions.

a) What pose corresponds to the identity transform?

b) What is the line `Pose_PoseInv.concat(invPose);` doing when the Pose_2D class does not even have a concat() method defined? Hint: Look up the java keyword `extends` and look for it in the Pose_2D file. Also look for `super.set(x,y,th);` and try to figure out what it is doing.

c) Would the following "pose composition" code work for an arbitrary pose in `Pose_PoseInv` and why?

```
Pose_2D invPose = Pose_PoseInv.inverse();
Pose_PoseInv.concat(invPose);
```

d) Unless you do abstract algebra for a living, look up the term "isomorphic". Are poses and *rigid* transforms isomorphic in 2D (i.e. x,y,theta)?

e) Look up "bijective" mapping. Are poses and *rigid* transforms isomorphic in 3D (i.e. x,t,z,theta,phi,psi)?

6. Converting Euler Angle Conventions (Extra Problem)

Suppose a pose box produces roll ϕ, pitch θ and yaw ψ according to the zyx Euler angle sequence. This means the composite rotation matrix to convert coordinates from the body frame to the world frame is given by:

$$R_b^w = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} = \begin{bmatrix} c\psi c\theta & (c\psi s\theta s\phi - s\psi c\phi) & (c\psi s\theta c\phi + s\psi s\phi) \\ s\psi c\theta & (s\psi s\theta s\phi + c\psi c\phi) & (s\psi s\theta c\phi - c\psi s\phi) \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}$$

The box is mounted on a vehicle with the y axis forward and the x axis to the right. Using the Euler angle outputs of the box in this configuration, compute the Euler angle outputs that would be generated if it were mounted with the x axis forward and y pointing to the left.

There are a few key issues here. First note that the 90 degree rotation is a rotation around the body z axis, not the world z axis around which yaw is measured. Second, note that the roll of b2 is approximately the pitch of b2 and the pitch of b2 is approximately the negative roll of b1. Unfortunately, the order of the rotations for the angles of b1 would be zxy if we simply swapped pitch and roll and swapped one sign. If this were done, the errors would go unnoticed except when both pitch and roll are of sufficient magnitude at the same time.