Introduction to Vision and Robotics Coursework 2019-2020

By Emma Hughes (S1802886) and Cezar Mihalcea (S1736865)

Github Repository Link

https://github.com/cezarmihalcea/IVR

To run the code, spawn the simulation in ROS and then create a new terminal and use `rosrun ivr_assignment image_angles.py` and then in another new terminal use `rosrun ivr_assignment controller.py`.

Vision

Discussion of Algorithms Used in Image Processing

For the joint state estimation, a trigonometric method was used: The position of the joints was calculated by threshlding their respective colours, based on the images obtained for each camera. Since the cameras are orthogonal, the coordinates were pooled together to form an array of 3d coordinates. Furthermore, in case a joint is not visible on one of the cameras, the coordinates were set to 0, and, for the z value, the maximum between the two camera values was taken.
Based on these coordinates, we calculated vectors inbetween each joint, as well as a [1, 0, 0] vector to determine the initial rotation of the yellow joint. The angles were computed using the following formula:

$$\cos(angle) = \frac{v1 \cdot v2}{||v1||||v2||}$$

Since the angles so computed are the geometric angles between the vectors, further operations were needed; namely, the angles were calculated in relation to their initial position (i.e. where they would be hand the joints not moved). The rotation was calculated by projecting the Blue-Green vector on the plane and computing the angle between [1,0,0] and this, then substracting the value of the third joint angle.

For target detection, first a thresholding algorithm was used to isolate the pixels with colour in the specific range of the circle and square. Thresholding involves creating a mask of the image where any pixel within a specified colour range – here, a shade of orange obtained by sampling – is set to 1 and all others are set to zero. Then a contouring algorithm was used to find the outline of each of the two orange blobs – by finding sharp gradient differences in the pixels, then the number of vertices of each of the two shape outlines detected was compared – the one with more vertices is the sphere. From here, a distance measurement was taken relative to the base of the robot for each axis by taking advantage of the multiple cameras, these distances converted from pixels to metres, and then the overall distance was calculated using Pythagorean theorem.

Accuracy of Target Position Tracking and Sources of Error

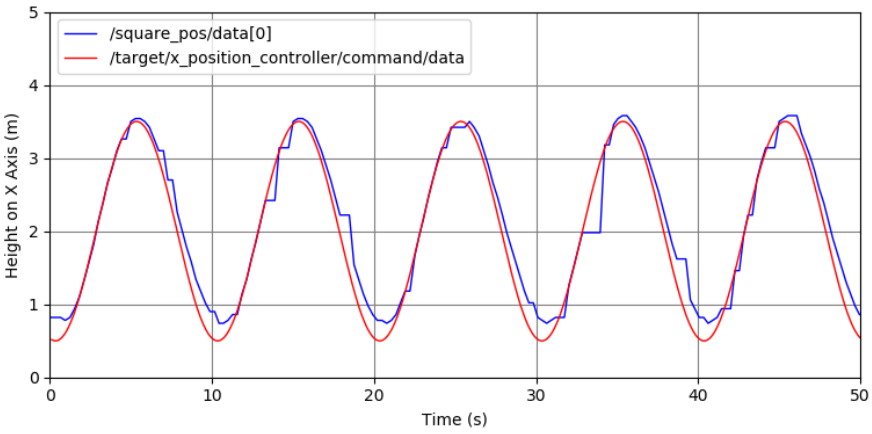
The angles were estimated to be as such, given 10 different positions:

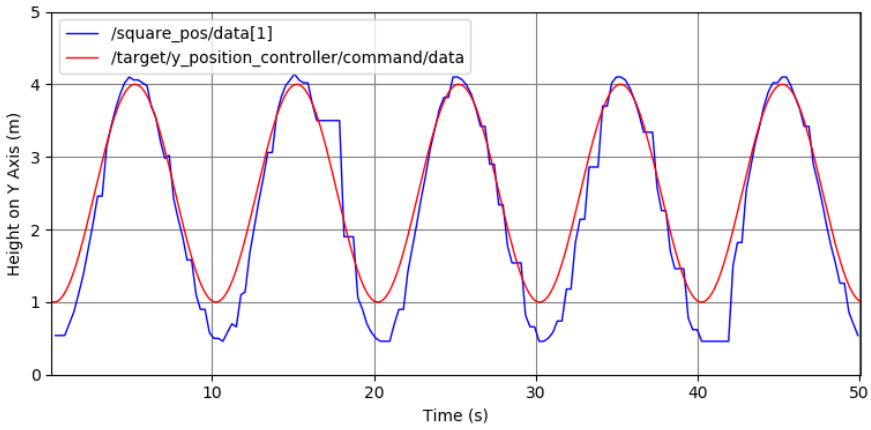
Joint1	joint2	joint3	joint4	estimated1	estimated2	estimated3	estimated4
0	0	0	0	0.46	0.12	0.02	0.01
0	0	1	1	0.05	0.165	1.14	0.88
0	1	0	1.57	2.99	0.96	0.07	1.57
0	1	1	1.57	0.89	1.26	1.19	1.57
1	1	1.57	1	0.004	1.26	1.19	0.92
1	1	-1	1	2.9	1.3	0.1	1.04

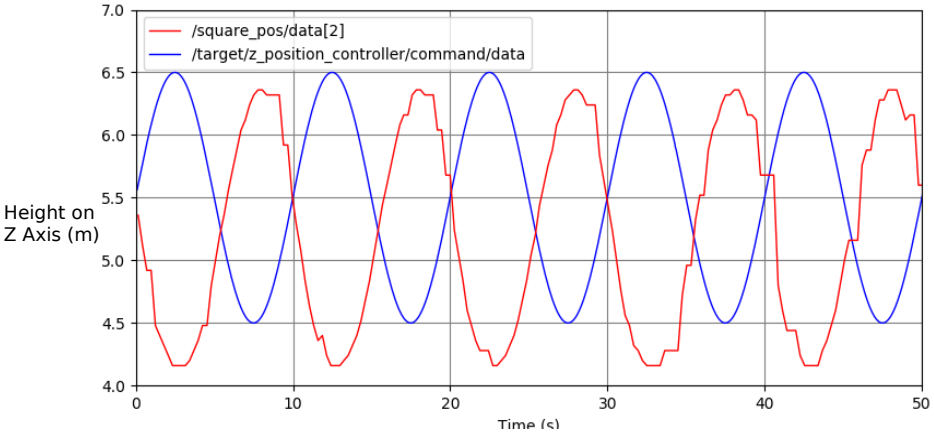
-2	1	0	0	0.52	0.32	0.85	0.28
2	-1	0	0	0.08	1.08	0.71	1.16
2	1	0	1	0.14	0.64	0.93	1.07
0	1	0	1	2.86	1.00	0.13	1.12

The estimations were, at large, erroneous. Most of the errors are due to the usage of the cosine to determine the angles. Since the cosine returns the same value for both positive and negative angles, the arccosine will always return a positive angle. Moreover, the algorithm would frequently confuse between a rotation of joint3 and a rotation of joint 1, resulting in further errors.

Please note that we were in fact tracking the sphere not the square but forgot to rename our topic accordingly.







One possible source of error in the measurements is the effect of perspective – as the sphere moves further away from the camera, perspective causes it to appear slightly closer to the horizon than it actually is, thus making the measurement on the z axis too small at distance, and too large up close.

Another source of error that was encountered was that occassionally the identification algorithm would fail to distinguish between the square and the circle when the angle on the square was particularly awkward and would return the position of the circle instead.
Also, the pixel to metre conversion method is not exact, and this can result in slighty erroneous results.
The pinhole camera may also produce a distorted, pinhole effect, causing the image to be contorted in a manner that is not true to life, causing erroneous readings.
It is as yet unknown why the z axis readings have effectively been inverted.

Robotics

Forward Kinematics Equation

$[x,y,z]^T = K(q)$

$q = [a,b,c,d]^T$

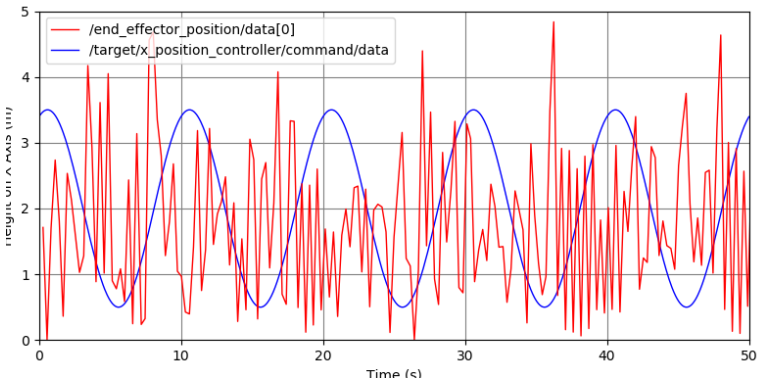
K = [(-3*cos(a+b+c)+3*cos(a-b+c)-3*cos(a+b-c)+3*cos(a-b-c)-2*cos(a+b+d)-2*cos(a-b+d)-cos(a+b+c+d)+cos(a-b+c+d)-cos(a+b-c+d)+cos(a-b-c+d)+2*cos(a+b-d)+2*cos(a-b-d)-cos(a+b+c-d)+cos(a-b+c-d)-cos(a+b-c-d)+cos(a-b-c-d)+6*sin(a+c)-6*sin(a-c)+2*sin(a+c+d)-2*sin(a-c+d)+2*sin(a+c-d)-2*sin(a-c-d))/4, (-6*cos(a+c)+6*cos(a-c)-2*cos(a+c+d)+2*cos(a-c+d)-2*cos(a+c-d)+2*cos(a-c-d)-3*sin(a+b+c)+3*sin(a-b+c)-3*sin(a+b-c)+3*sin(a-b-c)-2*sin(a+b+d)-2*sin(a-b+d)-sin(a+b+c+d)+sin(a-b+c+d)-sin(a+b-c+d)+sin(a-b-c+d)+2*sin(a+b-d)+2*sin(a-b-d)-sin(a+b+c-d)+sin(a-b+c-d)-sin(a+b-c-d)+sin(a-b-c-d))/4, (3*cos(b+c)+3*cos(b-c)+2*cos(b+d)+cos(b+c+d)+cos(b-c+d)-2*cos(b-d)+cos(b+c-d)+cos(b-c-d)+4)/2]^T

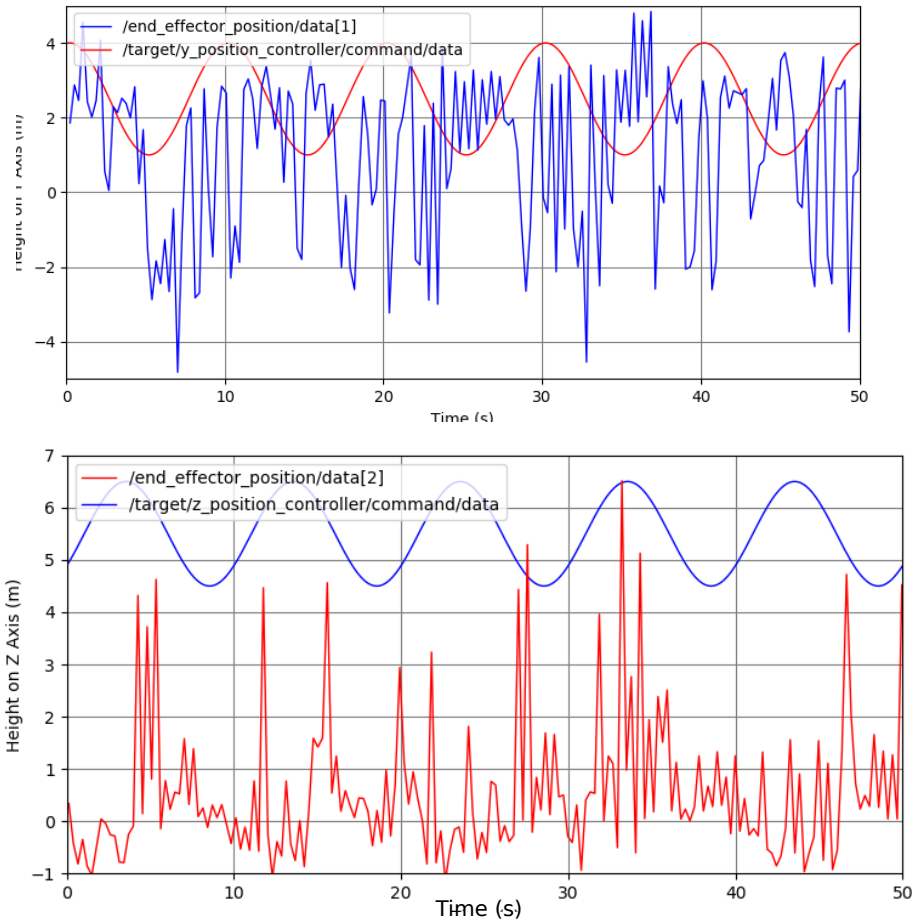
Accuracy of Forward Kinematics

The accuracy of the forward kinematics equation was tested by comparing the calculated values for 10 different sets of joint angles by eye to the actual position assumed by the robot when the robot was moved into the corresponding pose. All 10 tests yielded good results – as best as could be determined by eye, the end effector position was consistent with the coordinates provided.

Controller Accuracy and Sources of Error

Please note that we were in fact tracking the sphere not the square but forgot to rename our topic accordingly.





Possible sources of error include vision issues similar to those described in the sources of error for the target tracking – distortion due to perspective, distance, lenses, etc. - causing the joint measurements or target position tracking to be inaccurate. Notably, the joint angles measured in the vision component seem to have a large error attached to them, commonly up to 0.3 radians. This will impact on the robot's kinematics in turn.

Notably, this method relies on knowing the joint configuration of the robot – which cannot be determined uniquely from the pose of the robot – multiple joint settings could result in the exact same pose, entirely indistinguishable. Therefore, there is no guarantee that the joint angles measured were in fact the correct ones – they account for one possible configuration that would result in this pose. However, if this is not the correct configuration, then the inverse kinematics calculations could be adversely affected by this, even if the pose is the same. For example, the robot seems to struggle when negative angles are involved – the joint angle detection algorithm favours positive angles by default, whereas the inverse kinematic function has no such bias and will readily suggest negative angles, which the joint angle detection program will then inform it are in fact positive once it moves, resulting in an incorrect movement suggestion.

Also, the inverse kinematic calculation does not distinguish between solutions that are possible and those that are impossible due to the physical constraints of rotation – it will suggest a configuration that would work in theory, but in practise it may not be possible, which can lead to the robot attempting to assume a position that it cannot, and not reaching the target at all.

Also, the kinematic calculation does not have any lookahead to guess where the target will be once its movement is complete – the robot is constantly playing catch up and the refresh rate is much lower than for the target, plus it takes time for it to move its joints, so it is a few seconds behind at all times, meaning that it is not as close as it would be otherwise. It therefore over and undershoots the target a lot, which is why although the graphs do display that the robot follows the rough shape on most of the axises, there is a lot of noise.