EECS 206B Lab 1 Report: Trajectory Tracking with Baxter

Aakash Parikh, Federik Warburg, Jun Zeng

February 2019

1 Introduction

The overall goal of this lab was to determine our own trajectories for path planning and then use various controllers in order to allow Baxter to follow those paths. In this lab, we calculate 4 types of trajectories: a linear path that is a straight line to a goal, a circle around a target, a combination of linear paths and an AR tracking. We implement controllers in the joint space and workspace including PD velocity, torque and impedance controllers, and compare the performance to Baxter's built in functions for path planning and control in its MoveIt Package.

You can find a video of Baxter utilizing our controllers and our path planning here. and our GitHub repository here.

2 Path Infrastructure

In this first part of the lab, our team implemented three types of paths. In each of these, we look at the current location of the manipulator and the goal state (based on the AR tag) and calculate a path. For each time interval we have a target position, velocity and acceleration. We will later use these states for our controllers. The graphs of our target position, velocity and acceleration for each of our path planners can be found in Figures 1, 2, 3 in the appendix.

2.1 Line path

Our implementation of a straight line path to the target calculated the target positions, velocity, and acceleration of each of the x,y terms independently. We began by calculating our acceleration to traverse the desired distance in τ time. We required that our velocity begin and stop at 0. Thus we had a constant acceleration for the first half and a constant deceleration for the second half of the time period. Below, we shows the formulas to obtaining the acceleration, velocity and position of a line.

$$\tau(time for path) = d \cdot 5 \tag{1}$$

where d is the traversed distance (euclidean distance between target and current point). We found that using a constant 5 made the robot arm move at a desired velocity. We could increase/decrease this constant to make the robot move either faster or slower. We calculate the acceleration as

$$a(t) = \left\{ \begin{array}{ll} \frac{4d}{\tau^2}, & \text{for } t \leq \frac{\tau}{2}, \\ -\frac{4d}{\tau^2}, & \text{for } t > \frac{\tau}{2}, \end{array} \right\} \tag{2}$$

From this we can derive the velocity by solving:

$$v(t) = \left\{ \begin{array}{ll} a(t) \cdot t, & \text{for } t \leq \frac{\tau}{2}, \\ v(\frac{\tau}{2}) + a(t) \cdot (t - \frac{\tau}{2}), & \text{for } t > \frac{\tau}{2}, \end{array} \right\} \tag{3}$$

And the position:

$$p(t) = \left\{ \begin{array}{ll} \frac{1}{2} \cdot a(t) \cdot t^2 + a(t) \cdot p(0), & \text{for } t \leq \frac{\tau}{2}, \\ p(\frac{\tau}{2}) + \frac{1}{2} \cdot a(t) \cdot (t - \frac{\tau}{2})^2 + v(\frac{\tau}{2}) \cdot (t - \frac{\tau}{2}), & \text{for } t > \frac{\tau}{2}, \end{array} \right\} \tag{4}$$

The desired position, velocity and acceleration of this line path is presented in Figure 1 in the appendix.

2.2 Circular path

Another path we implemented was a circular path around a target AR tag. The manipulator should move in a circle and return in the same position. We began by calculating the angular kinematics for this, since we knew we had to complete a rotation of 2π radians in τ seconds. Similar to the linear path, we restricted our velocity to start and stop at 0, and our final position to be equal to our initial position.

$$\tau(time for path) = 2 \cdot \pi \cdot r \cdot 5 \tag{5}$$

We calculated the angular acceleration, velocity, and postion as

$$\alpha(t) = \left\{ \begin{array}{ll} \frac{8\pi}{\tau^2}, & \text{for } t \leq \frac{\tau}{2}, \\ -\frac{8\pi}{\tau^2}, & \text{for } t > \frac{\tau}{2}, \end{array} \right\} \tag{6}$$

$$\omega(t) = \left\{ \begin{array}{ll} \alpha(t) \cdot t, & \text{for } t \leq \frac{\tau}{2}, \\ \omega(\frac{\tau}{2}) + \alpha(t) \cdot (t - \frac{\tau}{2}), & \text{for } t > \frac{\tau}{2}, \end{array} \right\} \tag{7}$$

$$\theta(t) = \left\{ \begin{array}{ll} \frac{1}{2} \cdot \alpha(t) \cdot t^2 + \theta_0, & \text{for } t \leq \frac{\tau}{2}, \\ \theta(\frac{\tau}{2}) + \frac{1}{2} \cdot \alpha(t) \cdot (t - \frac{\tau}{2})^2 + \omega(\frac{\tau}{2}) \cdot (t - \frac{\tau}{2}) + \theta_0, & \text{for } t > \frac{\tau}{2}, \end{array} \right\} \tag{8}$$

where θ_0 was obtained by

$$\theta_0 = \pi - \tan^{-1}(current - goal) \tag{9}$$

We note that these are similarly defined as our kinematics equations from the linear path. However, we must convert these angular kinematics to euclidean linear kinematics. Let r (radius) be the euclidean distance between the initial position and the location of the AR tag.

$$a(t) = -r\omega(t)^2[-\alpha(t) \cdot sin(\theta) - \omega(t)^2 \cdot cos(\theta), \alpha(t) \cdot cos(\theta) - \omega(t)^2 \cdot sin(\theta), 0] \tag{10}$$

$$v(t) = r\omega(t)[-sin(\theta(t)), cos(\theta(t)), 0] \tag{11}$$

$$p(t) = r[cos(\theta(t) + \theta_0), sin(\theta(t) + \theta_0), 0] + p_d \tag{12}$$

The desired position, velocity and acceleration of this circular path around the target is presented in Figure 2, where a x-y plane view is also displayed.

2.3 Multiple path

Our square path is defined by 4 linear paths between each of the adjacent states: [current, goal(0), goal(1), goal(2), current]. Our implementation returns the kinematics of the current path, using the amount of time that we are on that path.

$$\texttt{pathtimes} = [\tau_{path_0}, \tau_{path_1}, \tau_{path_2}, \tau_{path_3}] \tag{13}$$

$$\texttt{currentPath}(t) = \min_i \sum_i \texttt{pathtimes}[: i] > t \tag{14}$$

$$\texttt{timeOnPath}(t) = t - \sum_{\texttt{currentPath}(t)} \texttt{pathtimes}[: \texttt{currentPath}(t)] \tag{15}$$

The illustration of desired position, velocity and acceleration is shown in Figure 3 in the appendix, where we can easily observe that it's a rectangle path.

3 Controller Implementation

In this section, we describe the 5 controllers that we implemented in order to execute our paths.

3.1 PD Workspace Velocity Controller

This controller has a proportional term about the end effector position error and a derivative term about the end effector velocity error. Precisely, the feed-forward PD controller can be described as below:

$$\dot{q}(t + \delta t) = \dot{q}(t) - k_p e_x - k_v e_{\dot{x}}, \tag{16}$$

where e_x represents the end effector position error which includes the translational error and the orientational error and $e_{\dot{x}}$ represents the end effector velocity errors. These errors are obtain in the joint spaces, and this we use the Jacobian of joint velocities to map to the workspace. We also need to be aware that the target orientation is not $[0,0,0]$. It should be the euler representation of the quaternion $[0,1,0,0]$ which is $[\pi,0,\pi]$.

3.2 PD Joint Space Velocity Controller

This controller is slightly different from the previous one, where the proportional term is about the joint angle value error and the derivative term is about the joint angular velocity error.

$$\dot{q}(t+\delta t) = \dot{q}(t) - k_p e_q - k_v e_{\dot{q}}, \tag{17}$$

where e_q represents the joint angle error and $e_{\dot{q}}$ represents the joint angular velocity error.

3.3 PD Joint Space Torque Controller

This controller is still a PD controller, however the output of this controller is the computed torque on each joint.

$$\tau = M\ddot{q}_d + C\dot{q} + N - k_p e_q - k_d e_{\dot{q}}, \tag{18}$$

where M, C and N represent the inertia matrix, coriolis matrix and gravity, respectively, and we use `inertia()`, `coriolis()` to get the inertia and coriolis matrix. The gravity matrix is obtained using $G = MJ^T[0,0,0.981,0,0]^T$. We use set `set_joint_torques()` to pass the computed torque values.

3.4 Workspace Impedance Controller

The impedance controller treats the external force as one of the variable to control the system. The workspace impedance controller can be written as,

$$F = F_{ext} + C\dot{x} + G + f + M\{\ddot{x}_d - M_d^{-1}[B_d e_{\dot{x}} + K_d e_x] + F_{ext}\} \tag{19}$$

$$\tau = J^T F, \tag{20}$$

where K_d and B_d represent the gains proportional (stiffness) and derivative (damping) term, while M_d represent the parameters for desired inertia matrix. In the general cases, F_{ext} should be measured from an external haptic sensor, we treat it as constantly zero since we don't apply external force on the end effector during the experiments. f is the friction force which is also regarded as zero during the implementation.

During the implementation, we have taken following simplifications:

- The Coriolis term in (19) is ignored, since it was not trivial to obtain it in workspace coordinates and neither seemed to have a big impact on the torque controller.
- The M_d is set to be equal to M, this simplification can help us to remove the term of F_{ext} in (19). Moreover this simplification makes us more easily to tune the parameters K_d and B_d.

Using these simplifications (19) is reduced to

$$F = G + M\ddot{x}_d - B_d e_{\dot{x}} + K_d e_x \tag{21}$$

which is very similar to the torque controllers. The advantage about using impedance controllers is that they allows you to model external forces. However, as the baxter robots do not have sensors to measure these external forces, it is reduced to a torque controller.

3.5 Joint Space Impedance Controller

Joint space impedance control is quite similar to the workspace one, where we replace the errors of the effector with the errors of joint angles:

$$F = F_{ext} + C\dot{x} + G + f + M\{\ddot{x}_d - M_d^{-1}[B_d e_{\dot{q}} + K_d e_q] + F_{ext}\}, \tag{22}$$

$$\tau = J^T F. \tag{23}$$

During the implementation, we have also set the M_d equal to M, this simplification can help us to remove the term of F_{ext} in (22). Moreover this simplification makes us more easily to tune the parameters K_d and B_d.

4 Visual Servoing

In this third part of the lab, we implemented visual servoing, where the goal is to follow an AR tag that change position. We implemented this in a sequential manner, where we made a slight modification of the `execute_path` method. Our goal for this, was to allow the robot to regularly query the location of the AR tag, and update its path when needed.

In particular we have enabled the following structure:

```
Result: Modification of execute_path method
Look up AR tag for initial destination;
Create a new LinearPath trajectory with initial destination;
while true do
    Use the current controller to execute the current path;
    if time is a multiple of 5 seconds then
        Look up AR tag for initial destination;
        if the AR Tag has moved then
            Create a new LinearPath trajectory with this new destination;
        end
    end
end
```

Algorithm 1: Visual Servoing Sequential Update

We have chosen to implement our method such that it only looks for AR tags every 5^{th} second as our `follow_ar_tag` method is slow (more than 1 second). In practice, slow performance of `follow_ar_tag` method results in an abrupt movement of the robot. This is slightly reduced by increasing the time between lookups, however, it comes at the cost of slower update of the AR positions. In future work, this method would be implemented in parallel such that the robot could look for new AR tags while moving smoothly.

5 Experiments and Results

5.1 Controller Tuning Process

For the PD Controllers, we use the standard PD tuning process described as below:

- Firstly, set the proportional and derivative term as zero and have a look at the feed-forward controller performance.
- Increase the proportional gain until the output of the loop oscillates. The proportional gain should be set to approximately half of that value for a ”quarter amplitude decay” type response. This needs an imprecise estimation.
- Be aware that for workspace controller, we tune position and orientation parameters respectively, while for joint space controller, we tune parameters for each joint sequentially. Once quasi optimal values are found, we could continue to use them in other similar controllers. You will observe the similarity in Table 1.
- Then we increase the derivative gain until the loop is acceptably quick to reach its reference after a load disturbance.

The performance won't be excellent as we don't have the integral term but this tuning process give quite a good performance which are well illustrated in our demo videos. In all tuning processes, we went through several iterations in order to observe the consequences of tuning parameters and obtain satisfactory results.

Based on the tuning process described as above, we also shared our parameters of each controller in Table 1.

Controller list	Proportional gain	Derivative term
Workspace velocity PD controller	$4 \times diag(1, 0.65, 0.1, 0.01, 0.01, 0.01)$	$0.07 \times diag(3, 0.5, 1, 1, 1, 1)$
Joint space velocity PD controller	$3 \times diag(1, 1, 1, 1, 1, 1, 1)$	$10^{-4} \times diag(1, 1, 1, 1, 1, 1, 1)$
Joint space torque PD controller	$8 \times diag(1, 1, 1.5, 1.5, 1, 1, 1)$	$5 \times diag(2, 2, 1, 1, 0.8, 0.3, 0.3)$
Workspace impedance controller	$0.5 \times diag(1, 1, 1, 0.2, 0.2, 0.2)$	$3 \times diag(15, 5, 9, 0.1, 0.1, 0.1)$
Joint space impedance controller	$8 \times diag(1, 1, 1.5, 1, 0.5, 0.5, 0.3)$	$3 \times diag(2, 2, 1, 1, 0.3, 0.3, 0.3)$

Table 1: Controllers Parameters

5.2 Controller Analysis

The convergent total of each controller strongly depend on the proportional and derivative parameters. Mathematically, if no disturbance and noise involved, we will observe the system converges faster with larger values of parameters. This property can be easily verified through the derivative of Lyapunov function. However we didn't observe this phenomena in the experiments, this might be due to the large noise of joint velocities.

The performance of our controllers can be compared directly through our demo. The desired and true end effector position and velocity as a function of time for each of the control methods and trajectories are plotted in the appendix, where performance of joint space velocity controller are shown in Figure 4, 5, 6, 7, workspace velocity controller in Figure 8, 9, 10, 11, torque controller in Figure 12, 13, 14, 15, joint space impedance controller in Figure 16, 17, 18, 19 and workspace impedance controller in Figure 20, 21, 22, 23.

Overall, we found mixed results in the performance of our controllers, which matches our analysis of the design of the controllers and the overall capabilities of Baxter.

- The performance of the first three controllers are quite good compared to Moveit, while the two impedance controllers are relatively worse than first three controllers. This might be due to to our numerical setting $M_d = M$.
- The joint space controllers performed slightly better than the workspace controllers. First, the interface of Baxter allows us to set torques and velocities in the joint space, which results in inaccuracies when having to convert workspace values to the joint space.
- We are able to fine tune the gains on each of the joints in a much more intuitive way for the joint space controllers. However, this does sometimes result in intermediate positions of the end effector not as tightly following the desired path in comparison to the work space controllers. This could potentially also lead to overfitting to a particular path, which we only briefly had time to investigate.
- The workspace impedance controller has issues when the Jacobian loses rank and we lose many degrees of freedom, sometimes rendering it unable to complete the task. The two impedance controllers are quite good on reaching the desired end effector position, while the orientation control performance needs more improvement.
- The derivative parameter of joint space velocity PD controller is quite small, we spent quite a lot time on tuning it. However, the joint space velocity observer is very noisy so we keep relatively small values of K_d for this controller.
- Baxter's built in MoveIt performs slightly better that our controllers in most cases. We believe this is due to the MoveIt controller having more information about the robot state and is better optimized for Baxter.

5.3 Applications of Controllers

The applications of controllers are predictable. The workspace and joint space controller can be applied for general applications, and are especially useful when the only sensors we have are for velocity and position. The torque controllers can be applied even without the speed sensors in the motors. In Baxter, we don't observe the advantages of this controller but this method could be useful for industrial robots. The impedance controllers are quite useful for robotics manipulation, especially for soft object manipulation, where the delicate external forces need to be observed. Moreover, the impedance controllers are also very useful for soft robots.

6 Conclusion

In this lab we implemented four different trajecoties namely Line Path, Circular Path, Multiple Path and AR tracking. We tested and compared the performance of multiple controllers that operated in both joint space and workspace. We compared our own implemented and fine tuned controllers with the Baxter MoveIt controller. We found that our jointspace controllers had slightly better performance than our workspace controllers. Especially, we obtained accurate trajectories using the PD jointspace controller and PD jointspace torque controller. After fine-tuning, these controllers obtained superior performance to the Baxter MoveIt controller.

7 Comments about lab documentation and starter code

Overall, it is a really nice lab, where we are introduced to many important practically concepts. We believe that the following would make it even better:

- Working helper functions (`cart_inertia` and `gravity`). The latter would could just be $G = MJ^T g$ where $g = [0, 0, .981, 0, 0, 0]$.
- A note on `numpy` shapes. We spent quite a long time using the default shaped arguments for the `step_control` which had the shape $(x,)$ rather than $(x, 1)$. As `numpy` do not know the orientation of these flat arrays and do not return error messages with matrix multiplication with quadratic matrices. We spent quite a long time debugging this error as expected the input to be correctly formatted.
- line 106 in `main.py` we changed to `path.to_robot_trajectory(num_way, controller_name!='workspace' and controller_name != 'workspace_impedance')` in order to use workspace for impedance.
- The impedance controllers in general seemed a bit off for this lab as the robot did not have any sensors to measure external forces, which (well-considered choices of constants) reduced the impedance controllers to torque controllers. These controllers might be omitted and instead more focus could be put on using an external camera sensor to find a more precise estimate of the AR positions. Alternatively, more focus could be put on methods to evaluate the performance of the sensors e.g. how precision estimation of camera estimate.

8 Appendix

8.1 Linear Path

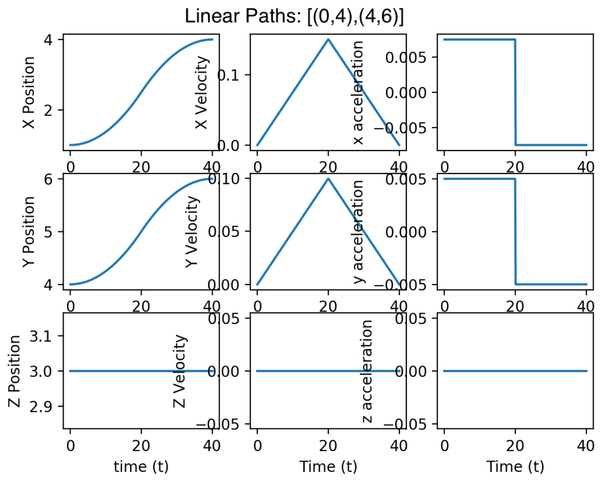


Figure 1: Linear path kinematics

8.2 Circular Path

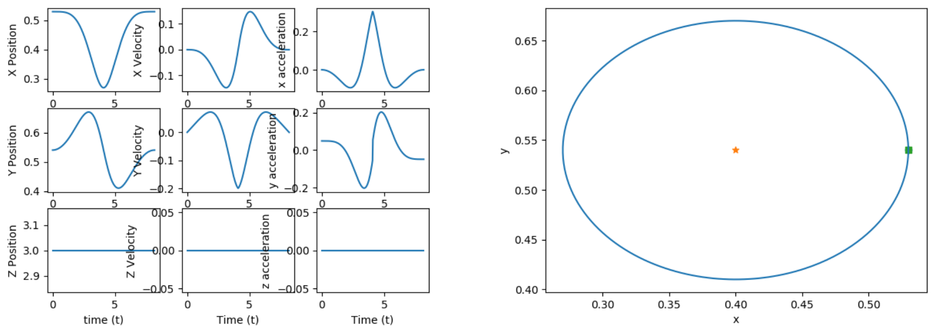


Figure 2: Circular path kinematics and a projective view from x-y plane

8.3 Multiple Paths

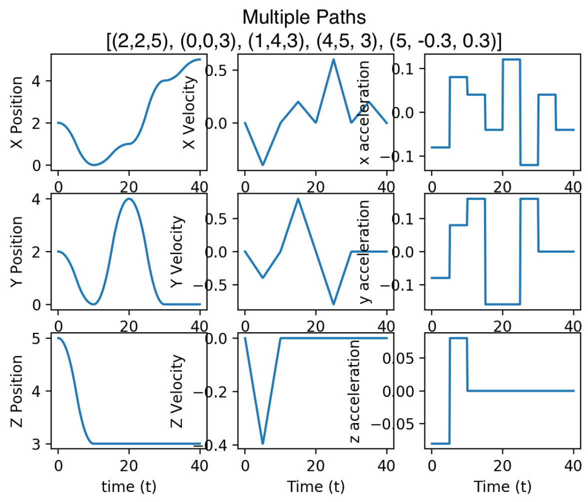
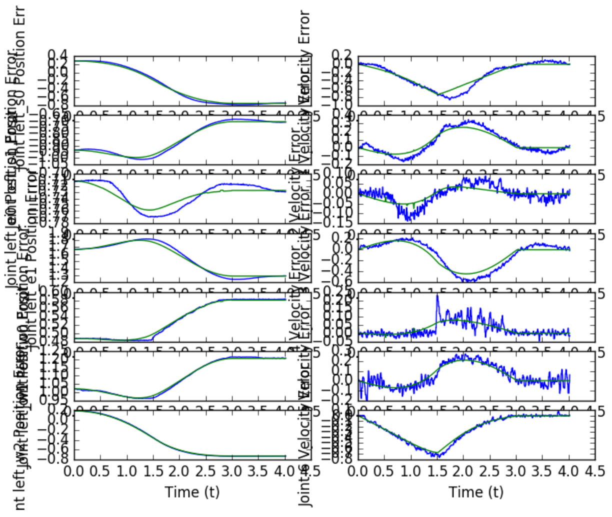


Figure 3: Multiple path kinematics

8.4 PD Jointspace Velocity Controller



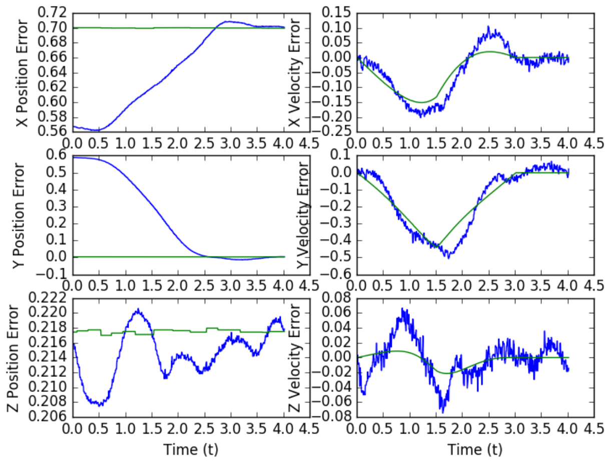
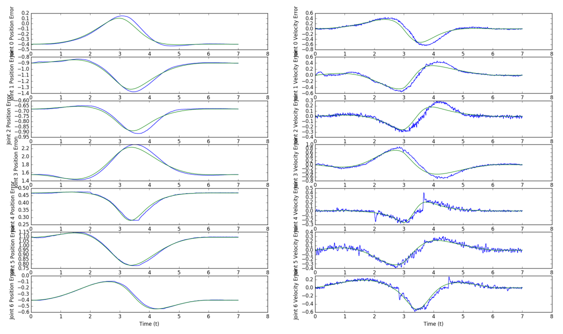


Figure 4: Joint space velocity controller to track the linear Path



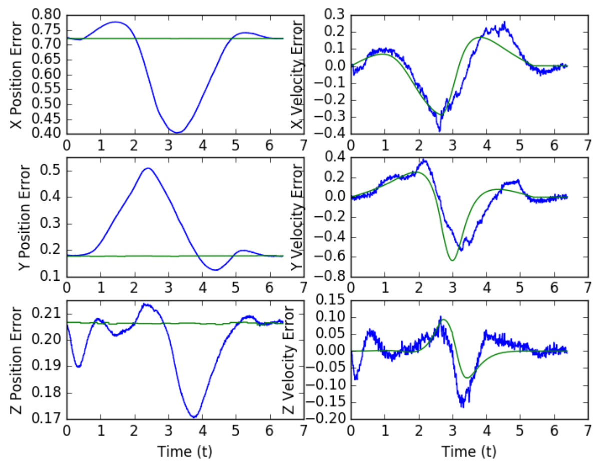
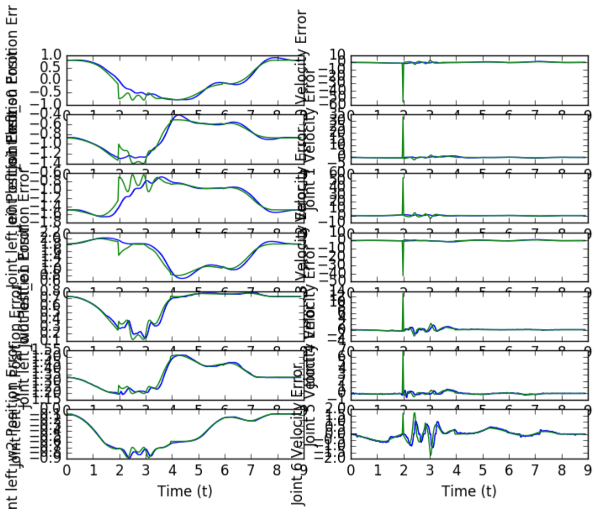


Figure 5: Joint space velocity controller to track the circular path



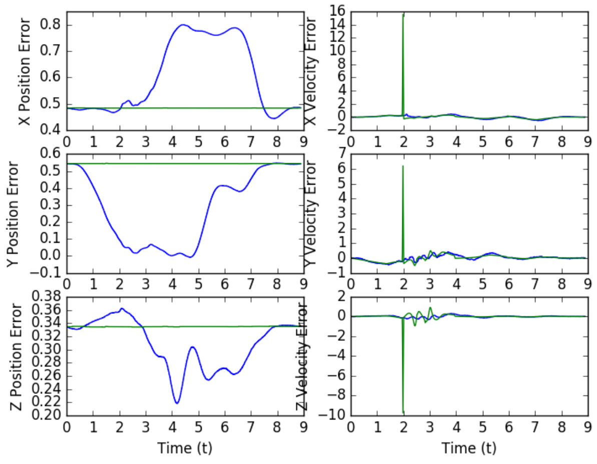


Figure 6: Joint space velocity controller to track multiple paths

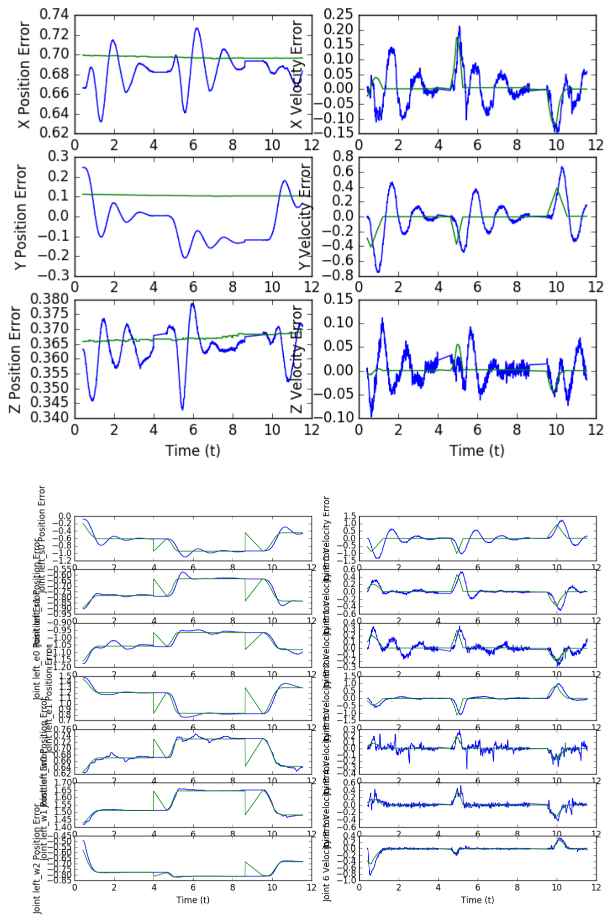


Figure 7: Joint space velocity controller to track the AR Marker

8.5 PD Workspace Velocity Controller

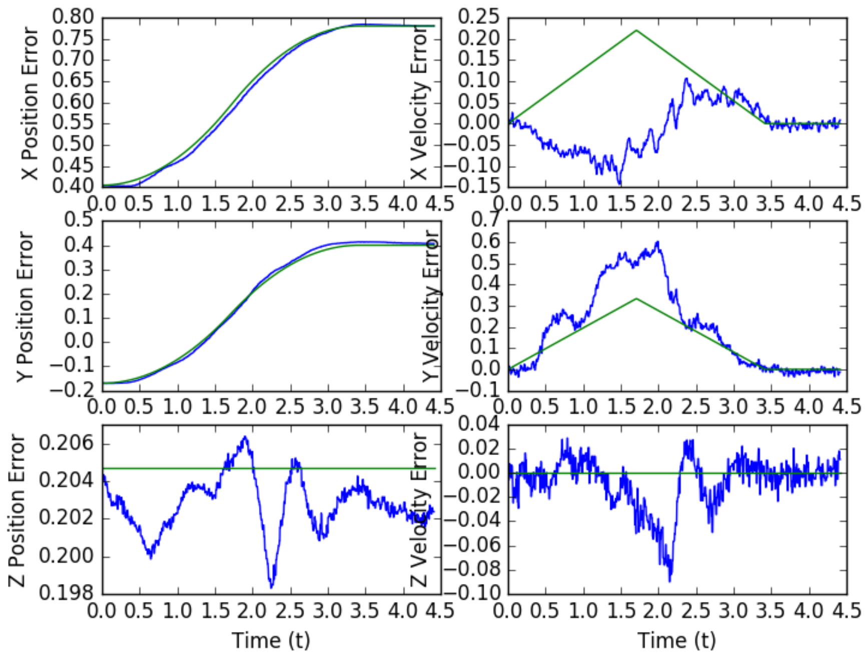


Figure 8: Workspace velocity controller to track the linear path

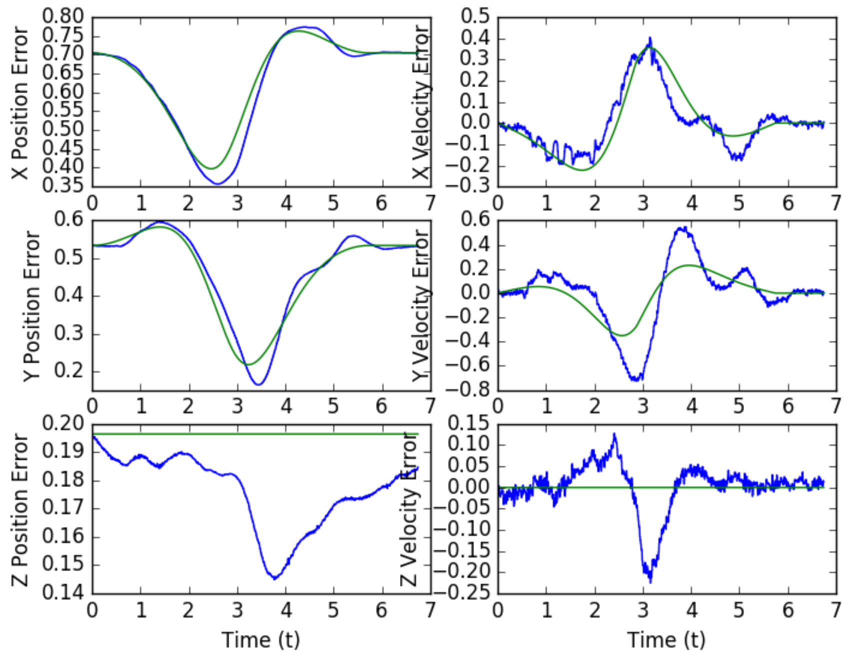


Figure 9: Workspace velocity controller to track the circular path

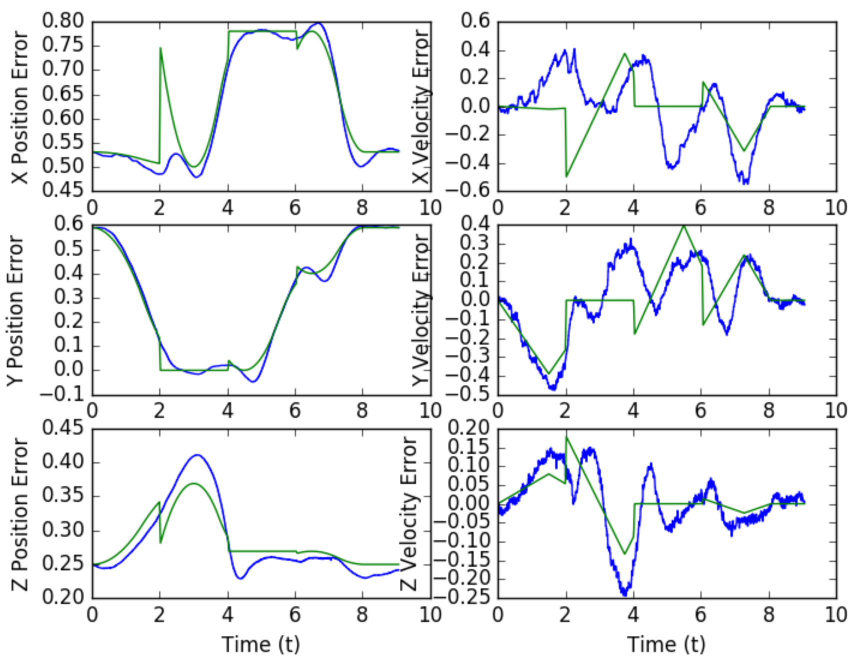


Figure 10: Workspace velocity controller to track the multiple path

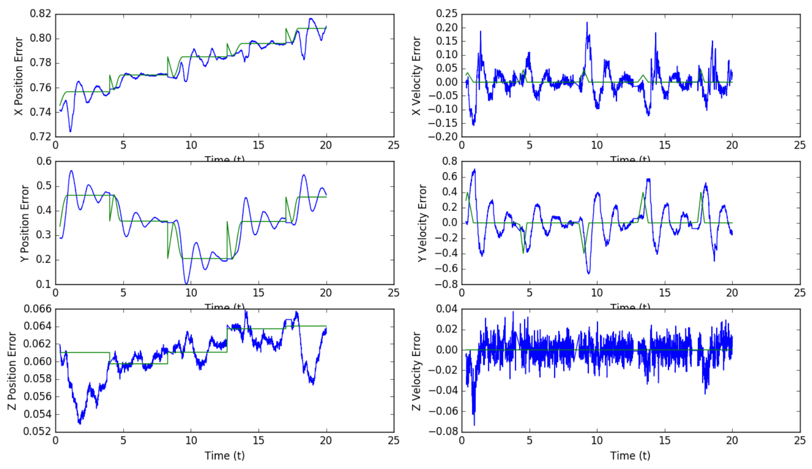
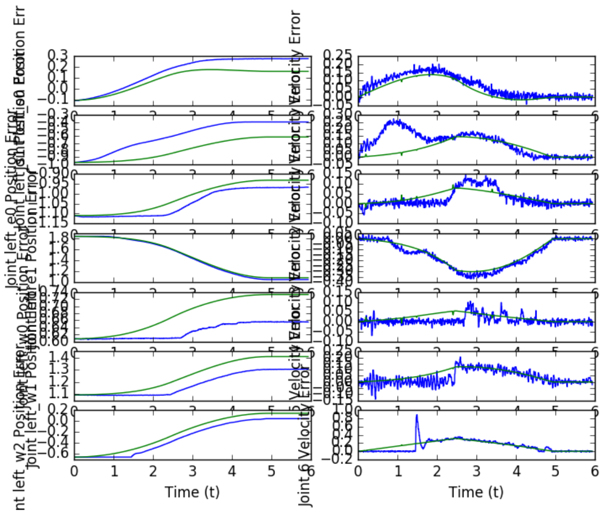


Figure 11: Workspace Velocity controller to track the AR Marker

8.6 Jointspace Torque Controller



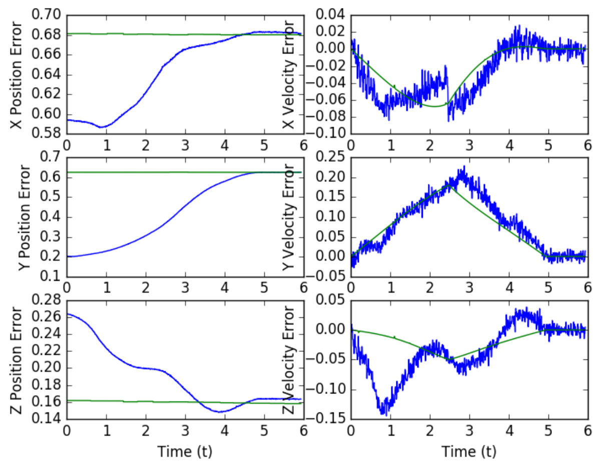


Figure 12: Torque controller to track the linear path

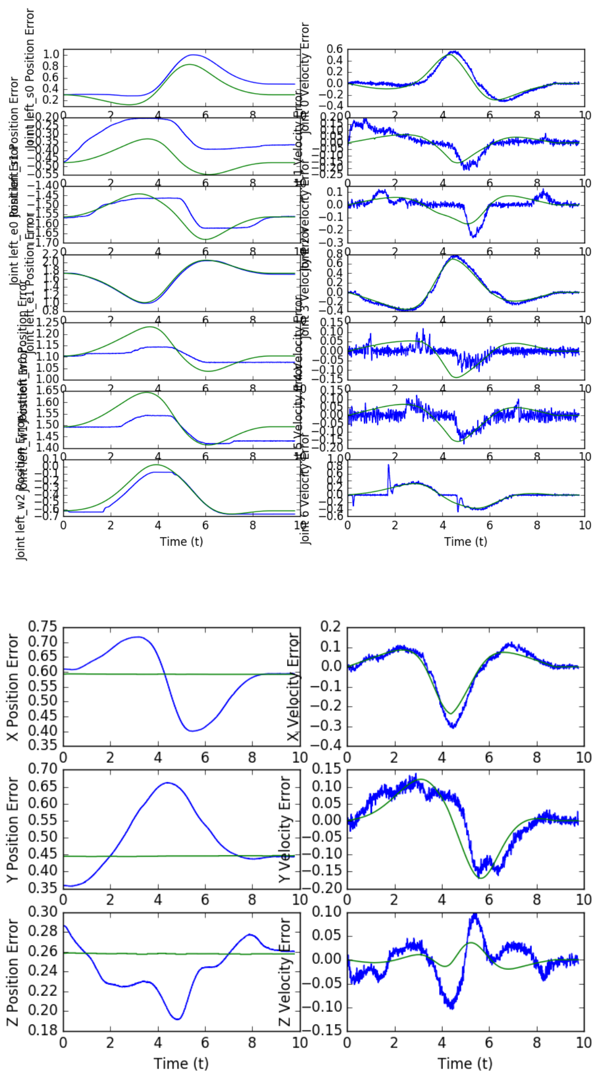


Figure 13: Torque controller to track the circular path

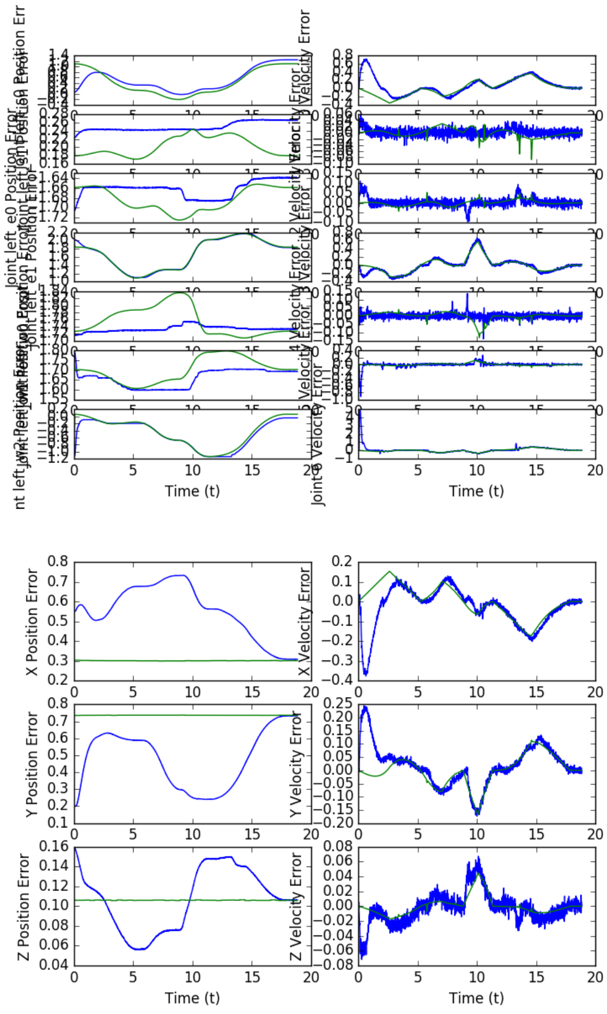


Figure 14: Torque controller to track the multiple path

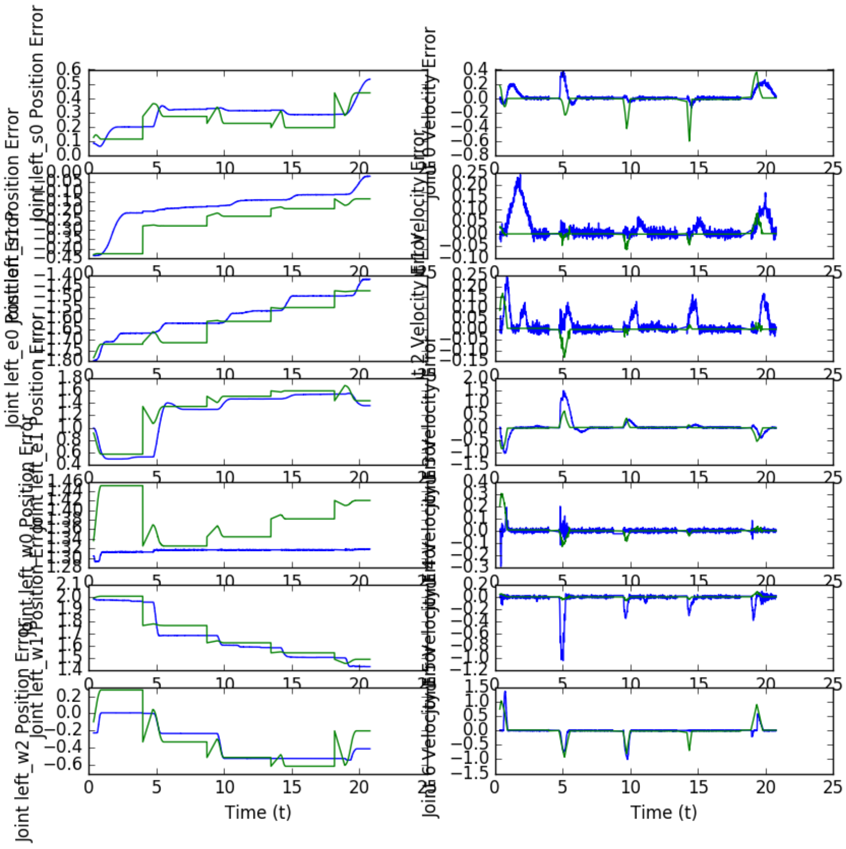


Figure 15: Torque controller to track AR Marker

8.7 Jointspace Impedance Controller

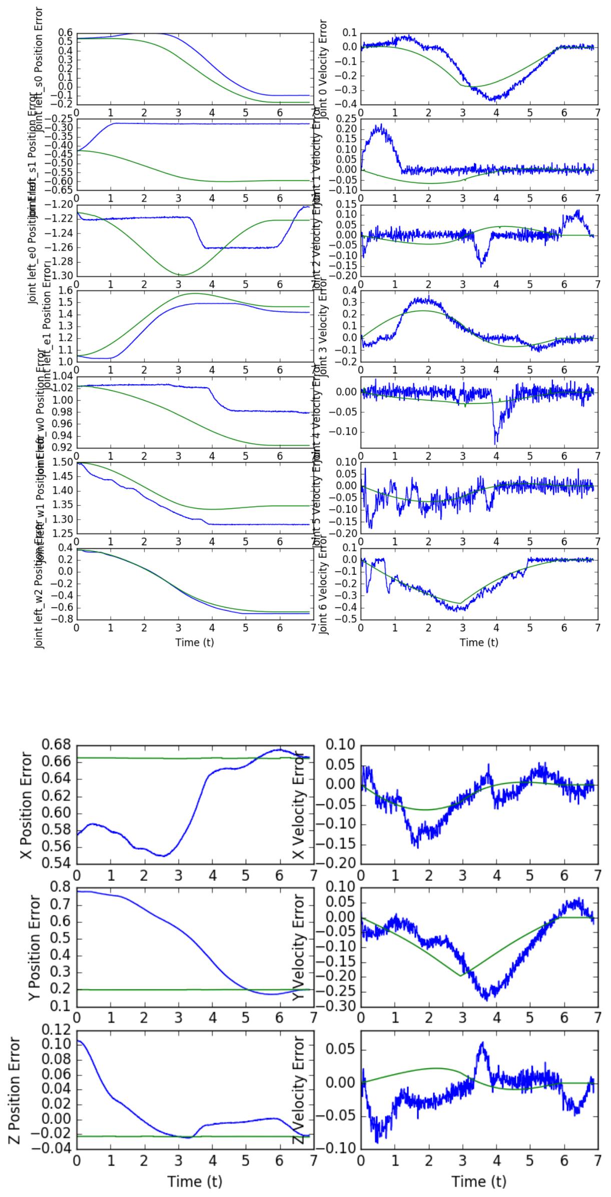
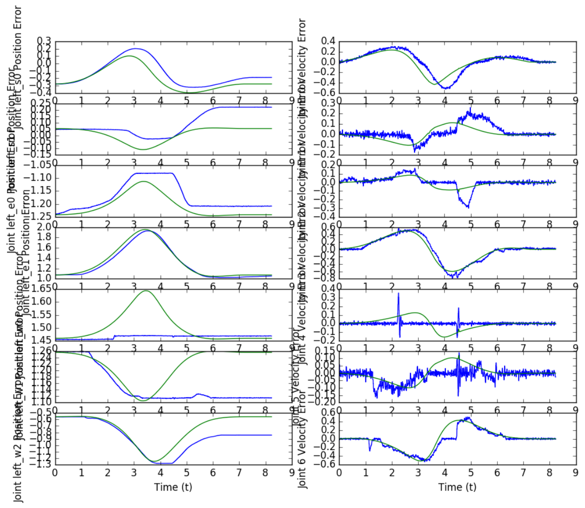


Figure 16: Joint space impedance controller to track the linear path



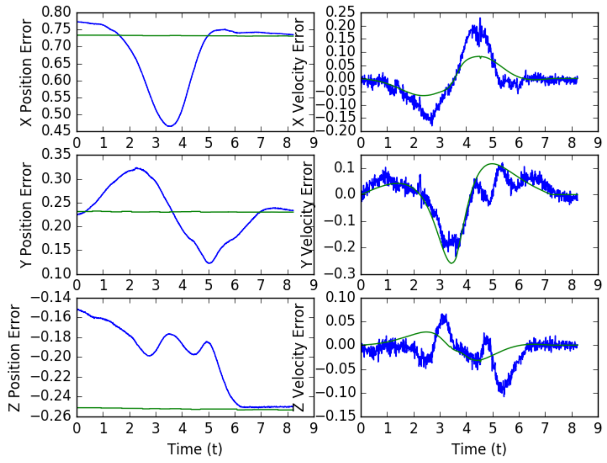


Figure 17: Joint space impedance controller to track the circular path

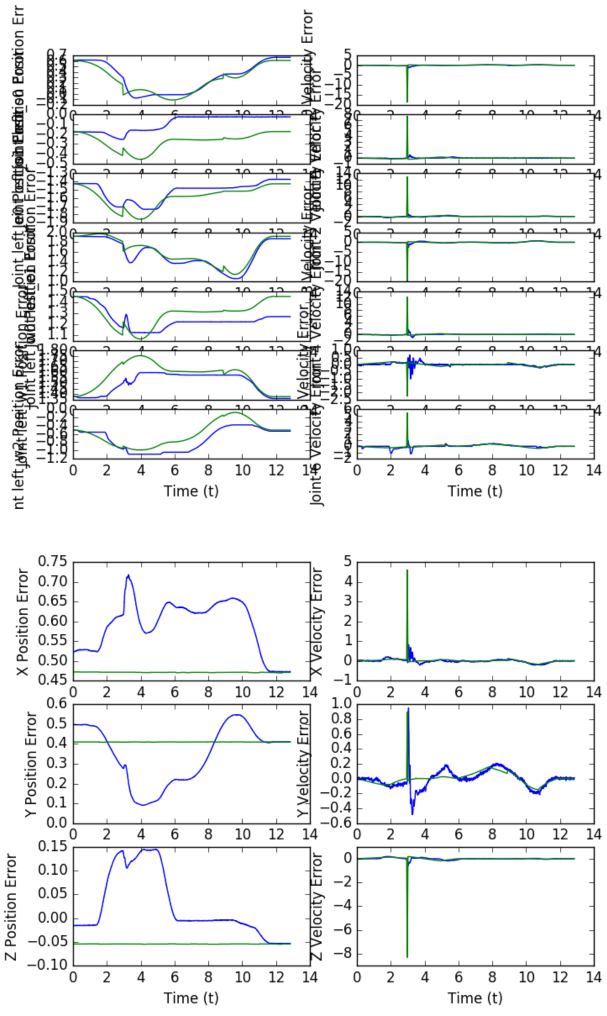


Figure 18: Joint space impedance controller to track the multiple path

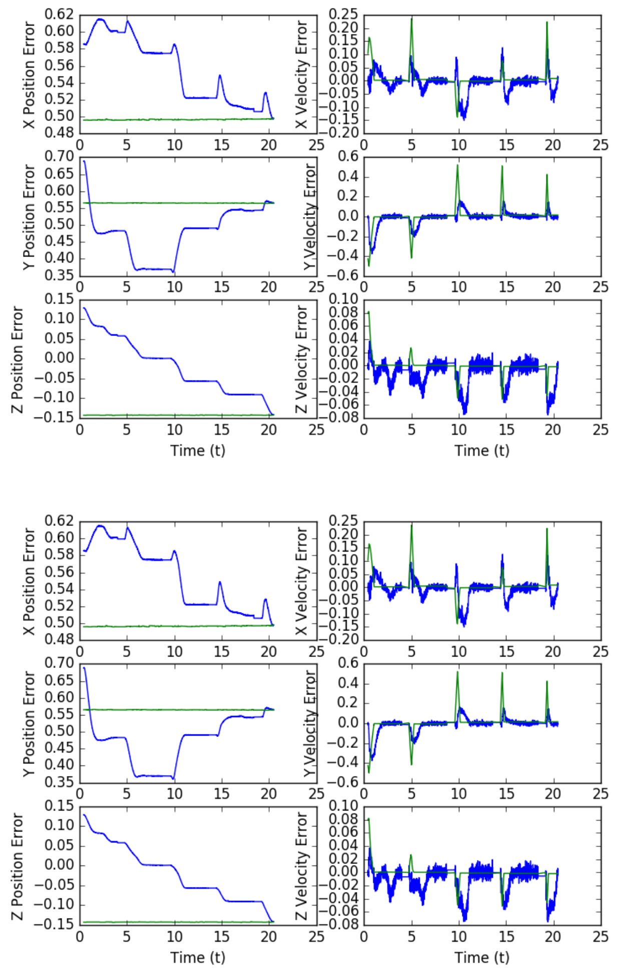


Figure 19: Joint space impedance controller to track the AR Marker

8.8 Workspace Impedance Controller

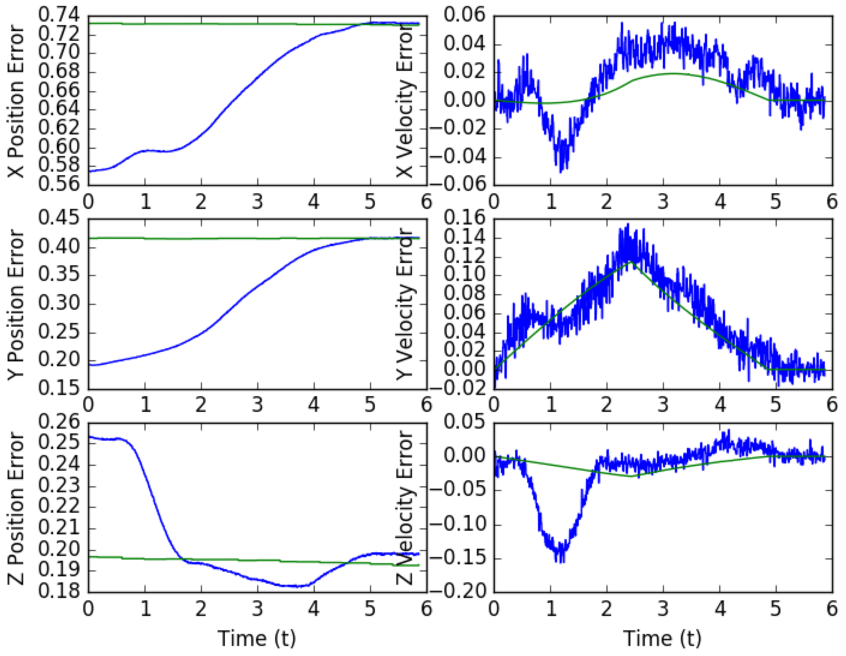


Figure 20: Workspace impedance controller to track the linear path

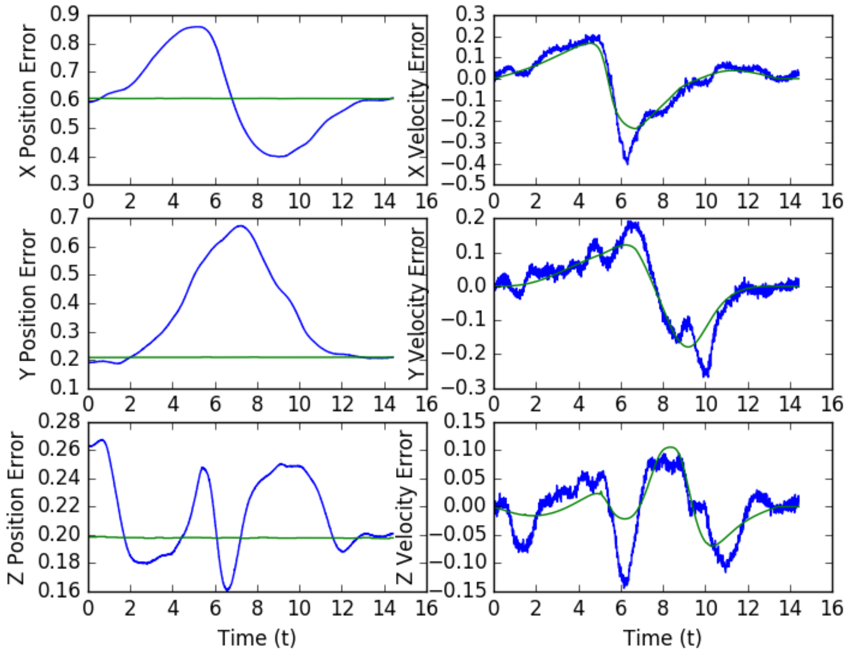


Figure 21: Workspace impedance controller to track the circular path

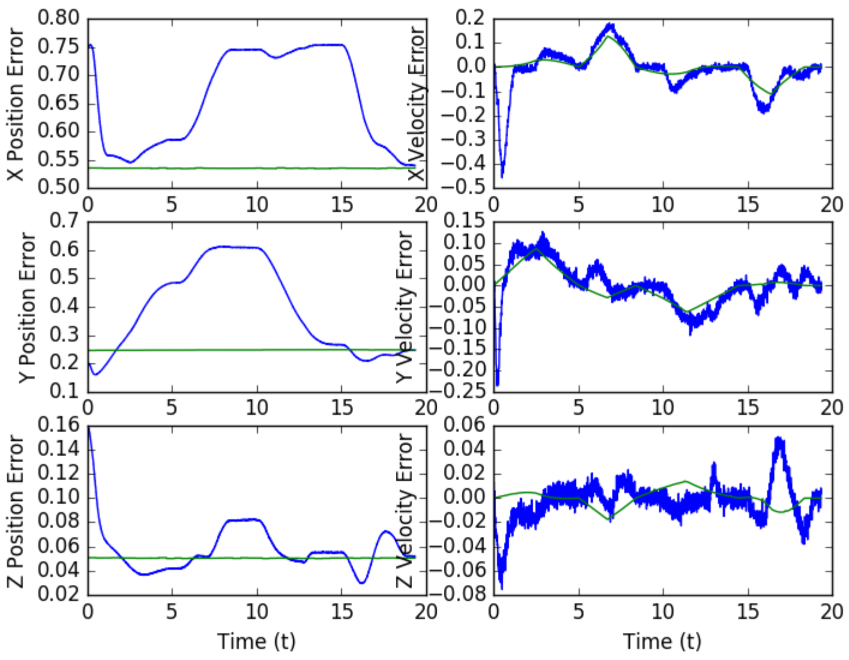


Figure 22: Work space impedance controller to track the multiple path

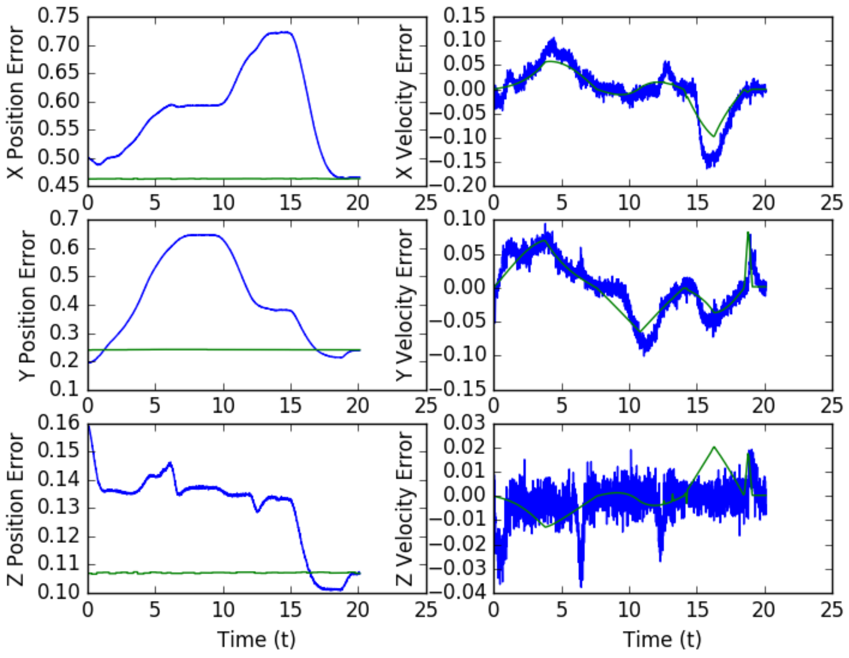


Figure 23: Workspace impedance controller to track the AR Marker