One-arm Juggling

6.4210/2 Robot Manipulation Project Final Report

Shao Yuan Chew Chia
Harvard University
Cambridge, MA
shaoyuan_chewchia@college.harvard.edu

David Jin
Massachusetts Institute of Technology
Cambridge, MA
jindavid@mit.edu

Richard Li
Massachusetts Institute of Technology
Cambridge, MA
mli97@mit.edu

***Abstract*—Picking and placing static objects with robot is becoming more common and vital in robotics research and even our daily lives. However, interacting with moving objects in a dynamical system is yet a difficult task as it requires accuracy in both space and time. This motivates us to work on such a course project that we design a system that allows a robot arm juggle. In this report, we present our methods and results of how we make the Kuka iiwa arm juggle with a squash-like rigid ball. We successfully let it juggle one ball to 2 meters high, and even juggle three balls, We also investigate the perception system that can help us with transferring from simulation to real world situation.**

I. Introduction

Seeing many examples of pick and place of static objects, we would like to design a robotic system that can interact with moving objects. We find the dynamic setting very challenging yet exciting. Compared to the static situation, this setting requires a higher level of the manipulation system and perception system. Therefore, Our goal is to make a Kuka iiwa arm juggle balls like a human juggler in simulation. By juggling we mean the gripper repeatedly catches (grasp), moves with (hold), and then re-throws (release) the ball(s), in a smooth motion. The arm has to match the velocity of the falling ball such that when catching the ball, from the gripper's frame, the ball is not moving. The gripper then smoothly transitions to the throwing velocity and release the ball at exactly the right time such that the trajectory of the ball follows a planned parabolic trajectory. We also include perception in our pipeline, using multiple Intel RealSense depth cameras. To initialize the juggling, we throw balls from a desired position with a desired initial velocity towards the gripper at the catch position. Due to gravity, balls in the air will inevitably fall to the ground, which creates a real-time constraint for our system. We carefully optimize the calculation of trajectory and arms/gripper poses so that the robot can catch the ball in time. Lastly, since we aim at juggling multiple balls, we design our trajectory optimization to update for each ball while tracking their order. We successfully make the robot arm juggles with at most three balls also at various heights.

II. Related Topics and Prior Works

A. Related Topics in Class

1) Object Motion planning: We perform object tracking in simulation to calculate the trajectory of the object being juggled. It is essential to model the trajectories correctly based on the properties of the object so that when we transfer to real word manipulation using the robot arm we don't have too large of a gap. Once we have the predicted position of the object at a specific time, we plan the motion of the gripper so that it can accurately maneuver in space to perform catching and throwing. We use inverse kinematics so that we can use the desired endofactor position to derive the motion of the arm from Chapter 8. We obtain the trajectory with kinematic trajectory optimization. We also use differential inverse kinematics to convert spatial velocities to joint velocities.

2) Perception: For transferring everything from simulation to real world juggling, we need to use perception through either depth cameras to track the object being juggled in real time with the knowledge from Chapter 4. There will be another round of tuning of how to deal with noise in the point cloud and how tracking of the object and motion planning will need to be done.

B. Prior Works

Our idea is similar to the previous year's course project of juggling a ping pong ball [1]. However, our project differs from theirs in two key ways: first, we want to catch, hold and re-throw the ball with the gripper instead of hitting the ball with the paddle; second, we want to juggle more than one ball; third, we want to incorporate perception and make it sufficiently robust such that we can juggle one ball on the real robot. Some previous related work built systems for throwing based on physical analysis by approximating the dynamics of objects [2] [3]. Those systems then tune the control parameters to control the trajectory. However, the real-world situation has plenty of factors that are not captured by the system, which leads to a low success rate.) Zeng et al. have designed a tossing bot that predicts the control parameters of the gripper via visual observation through deep learning [4]. By the learning approach, the tossing bot can decrease the error through implicit parameters in the model that were not accounted for in the dynamical systems.

Researchers have designed multiple different robotics systems that can juggle in some way, including but not limited to quadrotors [5], one-degree-of-freedom robots [6], and robotic arms [7] either independently as well as cooperatively. For multiple-object juggling, Rizzi and Koditschek created a robot

to work with multiple balls with a most simple juggling system, a ball, and a fixed planar paddle [8].

III. Experimental Setups

A. Manipulation Station

For our physical setup, The object we are juggling is a rigid ball that has the same dimensions as a squash ball. The arm we've chosen is the Kuka iiwa. It has a reach of 0.8 meters, a load capacity of 7kg, and 7 degrees of freedom. The gripper we're using is the WSG 2-finger gripper by Weiss Robotics. Notice that juggling with 2 fingers here is like trying to juggle with chopsticks. It'll force us to be really precise when throwing and catching.

B. Perception

We use three Intel RealSense depth cameras. Each of them produces a point cloud. We combine the point clouds, subtract the arm and then fit spheres to the point cloud to estimate the position of the ball.

IV. Methods

A. Ball Trajectory Calculation

1) Choosing the throw and catch positions: To start, we need to choose the points at which we are going to throw and catch. We attempt two approaches, the first is global inverse kinematics, where we fix the two x,y,z, spatial positions defined in the world frame, and then calculate what joint angles would lead to our gripper being at those positions, The second is forward kinematics where we first decide on a set of comfortable joint angles and then calculate where in space the gripper will end up being. Since we don't have specific requirements about where we need to throw and catch from, and what's really important is that we can comfortably throw and catch from those positions, the forward kinematics approach worked better for us. The following explanation describes what "comfortable" means for robot joints. Each joint has position limits if we add another joint, our configuration is a 2d shape. points outside this shape are not reachable. Given we have 7 joints, each joint adds another dimension on this graph, so we can try to imagine a 7-dimensional volume which gives all allowable sets of joint configurations. What we want is for the throw and catch positions to be well within that volume, not at the edges so that we aren't in any danger of needing our joint configuration to be somewhere outside the allowable range in order to reach a nearby spatial position.

2) Calculating throw velocity, catch velocity, and time duration: After determining the throw and catch positions $p^t = (p_x^t, p_y^t, p_z^t)^T$ and $p^c = (p_x^c, p_y^c, p_z^c)^T$ and their comfortable joint configurations, we use the formula of projectile motions to calculate what spatial velocities v^t the ball needs to have at the throw point to reach a given height h, as well as when the ball will pass through the catch point, and what velocity v^c it will have,

$$\begin{aligned} T &= \sqrt{\frac{2h}{g}} \\ v^t &= \begin{pmatrix} (p_x^c - p_x^t)/2T \\ (p_y^c - p_y^t)/2T \\ gT \end{pmatrix} \\ v^c &= \begin{pmatrix} (p_x^c - p_x^t)/2T \\ (p_y^c - p_y^t)/2T \\ -gT \end{pmatrix}, \end{aligned} \tag{1}$$

where $g = 9.81 m/s^2$ is the gavitational constant. Then, we also want to get the corresponding joint velocity. We first calculate the Jacobian matrix J of translational velocity at the joint configuration. By multiplying the pseudoinverse of the Jacobian matrix J with the spatial velocities v, we are able to obtain the joint velocities V for both catch and throw,

$$V = J^{\dagger} v.$$

The reason why we use pseudoinverse but not regular inverse is that the Jacobian matrix J is not a square matrix as the dimension of spatial velocities is 3 and the dimension of joint velocities is 7.

B. Kinematic Trajectory Optimization

1) Initialization: We use kinematic trajectory optimization to find a trajectory that satisfies multiple constraints for the robot arm. We want to minimize the length of the path, subject to the following constraints: first we constrain the positions and velocities at the throw and catch points. Second, the entire duration of the arm trajectory needs to be exactly the duration the ball is in the air for, and third we ensure that the joints on the iiwa robot don't exceed their position, velocity, or acceleration limits. Here is the optimization problem of the trajectory between throw and catch with N control points:

$$\begin{aligned} \min_{q_0, q_1, \ldots, q_N} \quad & \sum_{n=0}^{N} ||q_{n+1} - q_n||_2^2 \\ \text{s.t.} \quad & q_0 = q_{\text{throw}} \\ & q_N = q_{\text{catch}} \\ & v_0 = v_{\text{throw}} \\ & v_N = v_{\text{catch}} \\ & q_{\text{lower}} \le q \le q_{\text{upper}} \\ & v_{\text{lower}} \le v \le v_{\text{upper}} \\ & a_{\text{lower}} \le a \le a_{\text{upper}} \\ & t_N - t_0 = T \end{aligned} \tag{2}$$

2) Update: We not only use kinematic trajectory optimization when initializing the trajectory for the robot arm but we also use it for updating the new trajectory every time after it throws. This update is necessary because, without such an update, the gripper is trying to catch the ball where it previously thinks where the ball is from the initialization

instead of where the ball actually is. To account for this we change our planner so that it updates the trajectory based on the actual throw position and velocity right after it lets go of the ball. Given the actual position and velocity at the throw point, we can calculate where the ball will be a certain duration later. Then, we use the differential inverse kinematic to find the positions and velocities in the joint space before sending to the update of kinematic trajectory optimization.

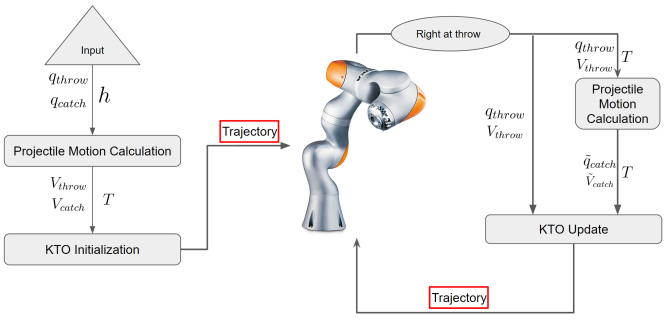

Fig. 1. Diagram of kinematic trajectory optimization (KTO) initialization and update: The inputs are thrown position q_{throw}, catch position q_{catch}, and the max height h of the projectile motion of the ball. Then, we obtain throw velocity V_{throw}, catch velocity V_{catch}, and time duration T for the projectile. When update, we get the actual catch position and velocity as $\tilde{q}_{catch}$ and $\tilde{V}_{catch}$.

C. Differential Inverse Kinematic

For converting spatial velocities/positions into joint space, we think of two methods: the first is to apply spatial position and velocity constraints at the predicted catch point, and the second is to convert the spatial position to joint position using differential inverse kinematics. The first one does not work because our solver could not find solutions given all position and velocity constraints are in joint space. We then fall back to the other method in which we stick to only giving it joint position and velocity constraints. Due to the fact that there could be multiple joint configurations for one spatial position, we could not use inverse kinematics to convert from spatial space to joint space. Thus, we use differential inverse kinematics to find the joint positions and velocities.

There are two difficulties when applying this method. First, as discussed earlier it might be near the boundary of the volume that defines permissible configurations. Second, our throw and catch joint configurations were chosen particularly to be very close together in configuration space, and an arbitrary solution would probably be further away in configuration space meaning the arm has to move more to reach that particular configuration, which might exceed the time the robot needs to maneuver. Using differential inverse kinematics, we start from our comfortable joint configuration and iteratively nudge the joints such that it gets closer and closer to our desired spatial position. Through this way, we know that the two configurations will be near each other and only a relatively small adjustment needs to be made in the trajectory.

D. Inverse Dynamics Controller

Due to the precision that we need in order to catch and throw the ball at a very specific position, velocity and time, any small delay will end up in a bigger error that leads to a failure to juggle. To solve this problem we need to peel back the layers of the onion and see how the commands we are sending to the iiwa actually get translated into the joints moving. Originally we are sending our desired positions to a system that interpolates the desired state with discrete derivatives. It takes discrete derivatives of the positions you send it to generate velocities, and the positions and velocities together make up the state. The desired state is then fed into the inverse dynamics controller which spits out the force commands to send to the arm. To increase the accuracy, and minimize the delay, we cut out the state interpolator and take analytical derivatives on our position b-spline trajectory to get our desired velocities and accelerations and feed them into the inverse dynamics controller directly. While the original controller was error based, meaning it only makes corrections when there is a deviation between desired and measured values For the new controller, once it's on the trajectory, it will be given the force command it needs to stay on the trajectory

E. Perception

We needed a perception system to integrate the robot juggling system into a real iiwa robot since we will need to measure the real-time position and velocity of the ball to account for real-world perturbation. We would then be able to use the position and time difference between two adjacent points on the trajectory to estimate the velocity and predict future trajectory to optimize the gripper pose.

We can first generate point clouds from the depth and RGB images using used three Intel Real-sense cameras. We then take the average of the X, Y, and Z positions of the cloud to estimate the center position of the ball. Shortly after we realized that for this approach we would be getting a point cloud with not only the ball but also the gripper as well, which doesn't allow us to accurately predict the center of the ball. So we thought of other different ways to remove the gripper from the point cloud:

1) Training a neural net of the gripper and the ball to create labeled images
2) Obtain the collision geometry of the arm and gripper to subtract from the concatenated point cloud
3) Obtain the labeled image from the manipulation station and use the masked image to crop out everything but the ball

We chose the third approach. We first obtain the masked image of the ball, and get the pixels from the masked image, then match the corresponding pixels with the depth image to generate an image of the ball with the depth data. We can then generate a point cloud from this image. However, we still can't simply take an average of this point cloud, as there would only be a partial point cloud of the ball left due to the gripper blocking the camera view. To overcome this, we

decided to fit a sphere to the point cloud to find the center and radius of the ball through least square optimization. This way we can still find the center position even if we only have a partial point cloud of the ball.

V. Result and Discussion

A. One ball Juggling

At first, we are having trouble making the robot juggles one ball. The two major difficulties are

1) the actual velocities of the gripper are different from we plan according to the kinematic trajectory optimization, and
2) actual catch position is different from the planned catch position.

We use inputting acceleration to the inverse dynamics controller to solve the first one and update the kinematic trajectory optimization after the throw to solve the other one. We also test the one ball juggling successfully with various heights, e.g. 0.2m, 0.75m, and even 2m.

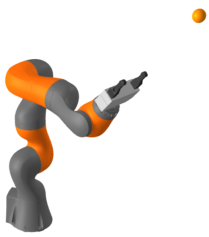

Fig. 2. A screenshot of throwing when juggling one ball at 2m

Fig. 3. A screenshot of catching when juggling one ball at 2m

B. Two ball Juggling

To make the jump to juggling multiple balls we modify our planner for updating the trajectory. When trying to juggle two balls, after throwing the first ball, it has to catch and throw the second ball, all in the time that the first ball is in the air. Thus, there is much less time to move between the throw and catch positions. We can keep most of our planner exactly the same, but after a ball is thrown, instead of re-planning the trajectory based on the ball we just threw, we update it with the position and velocity of the ball that we want to catch next. And this is what it looks like

We also include a warm start of the arm with the desired joint velocities at time 0 because, unlike the one-ball case, it does not have enough time for the actual velocities to converge on the desired velocities before it has to throw the ball.

Fig. 4. A screenshot of juggling two balls

C. Three ball Juggling

Next, we try juggling 3 balls, but keep running into the problem of the balls colliding in the air with each other because now there are two balls in the air at the same time and slight inaccuracies in the throw. One reason is that the trajectory of the projectile motion of the ball we plan is narrow due to the close throw and catch points. Then, the balls will collide due to their radius being larger than half of the trajectory at their height. The other reason is that juggling three balls requires a longer time, thus they are thrown much higher than before. This leads to a significantly faster throw and catches velocity which is hard for our iiwa arm to achieve.

Fig. 5. A screenshot of juggling three balls

We successfully juggle three balls by decreasing the radius of the balls. Though we do not figure out how to juggle with squash balls, it shows the ability of our robot arm to catch tiny objects. Unfortunately, we are not able to figure out how to juggle four balls as the kinematic trajectory optimization fails every time.

D. Perception

We get the depth camera connected to the manipulation station and are able to produce a visual representation of the point cloud when the ball is falling. We also write an algorithm for fitting a sphere to the particle point cloud. Though encountered some difficulties in subtracting the gripper and robot arm, we access the labeled image port and obtained the masked image of the ball with the gripper removed. However, by the time of this report, we have not been able to correlate the pixel data with the depth data, thus can't generate the partial point cloud of the ball for us to fit a sphere.

Fig. 6. Masked images of the free falling ball at time step 1 (left), time step 2 (middle) and time step 3 (right). These images show how the ball enters the view of the camera and then leaves.

VI. Conclusion and Future Work

We successfully make the iiwa arm juggles with balls. Our project allows it to juggle one ball with a variety of heights and even juggle three balls at a very fast speed. We demonstrate the ability of the robot to catch and throw very tiny objects.

For future work, we would like to further improve the system to have the robot juggling more balls, or even objects with different shapes, where we need to consider the orientation of the object for selecting the grasp. We would also like to give the robot additional flexibility in choosing when to catch, instead of constraining the time duration between catch and throw. Furthermore, to integrate the system into the real robot, we would need to finish our work in perception and then develop a controller that's robust enough against camera noise and real-world disturbances that might change the ball trajectory.

Contributions

Shao contributes to the experimental setup, manipulation, and juggling with one and multiple balls. David contributes to ball motion calculation, manipulation (trajectory optimization and inverse dynamics controller), and juggling with one ball. Richard contributes to the experimental setup, perception, and juggling with noise and perturbation.

Acknowledgment

We would like to thank Professor Russ Tedrake for his help and the wonderful lectures he gave during this semester. We also greatly thank all the TAs in the class, who are always very helpful and supportive.

References

[1] R. Shubert, M. Beveridge, "Robot Juggler", https://youtu.be/m5UnMWihWC4, 2020.

[2] W. Mori, J. Ueda, and T. Ogasawara, "1-dof dynamic pitching robot that independently controls velocity, angular velocity, and direction of a ball: Contact models and motion planning," IEEE International Conference on Robotics and Automation, 2009.

[3] T. Senoo, A. Namiki, and M. Ishikawa, "High-speed throwing motion based on kinetic chain approach," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.

[4] A. Zeng, S. Song, J. Lee, and A. Rodriguez, T. Funkhouser, "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics," IEEE Transactions on Robotics, 2020.

[5] W. Dong, G.-Y. Gu, Y. Ding, X. Zhu, and H. Ding, "Ball juggling with an under-actuated flying robot," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2015.

[6] A. Zavala-Rio and B. Brogliato, "On the control of a one degree-of-freedom juggling robot," Dynamics and Control, 1999.

[7] D. Serra, F. Ruggiero, V. Lippiello, and B. Siciliano, "A nonlinear least squares approach for nonprehensile dual-hand robotic ball juggling," IFAC-PapersOnLine, 2017.

[8] A. A. Rizzi and D. E. Koditschek, "Further progress in robot juggling: The spatial two-juggle," IEEE International Conference on Robotics and Automation, 1993.

Appendix

The appendix shows the desired positions and velocities (blue) compared to the actual positions and velocities (orange) of all 7 joints when juggling with three balls. It demonstrates that our controller can accurately control the robot to achieve the desired position and velocities.s

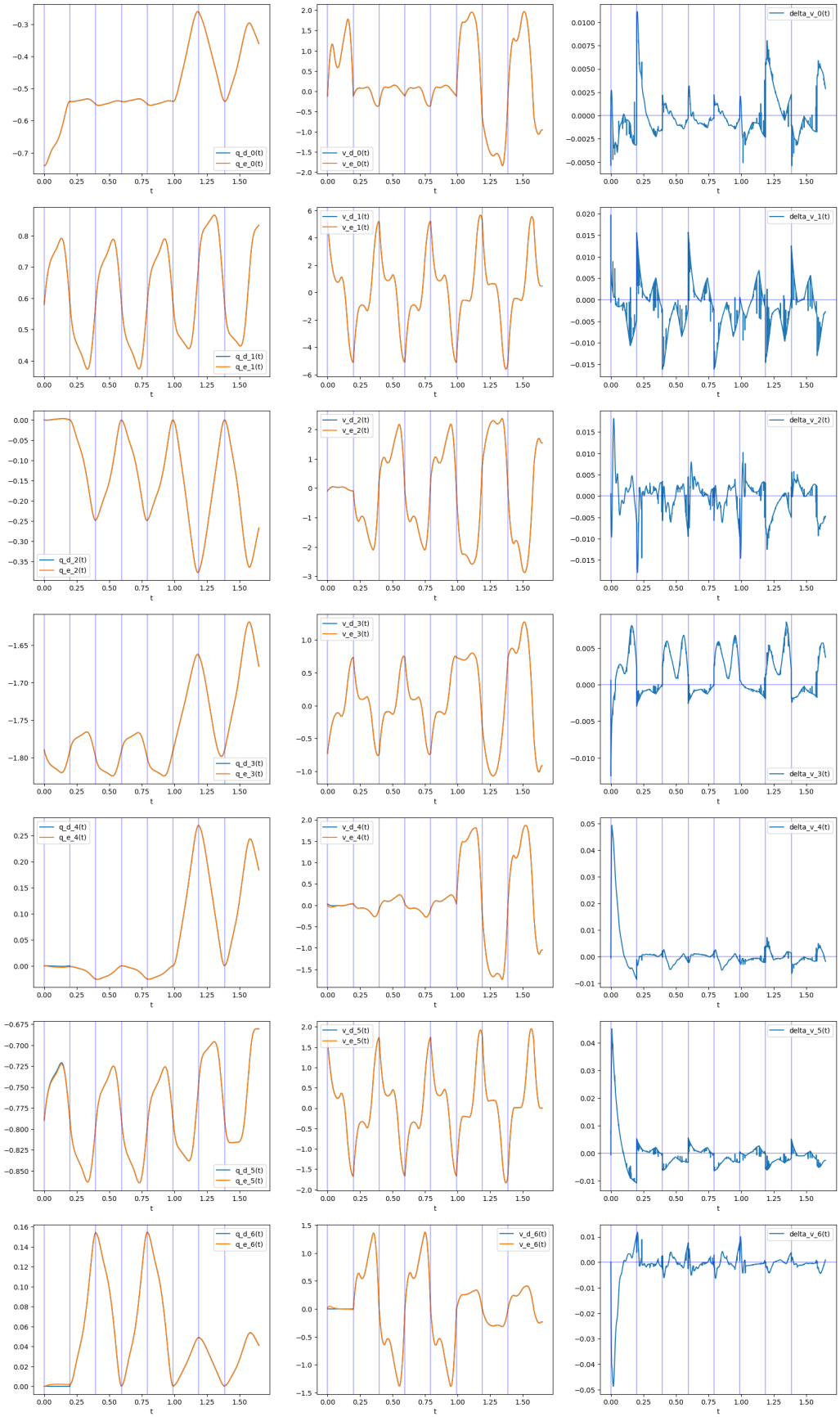

Fig. 7. Desired vs Actual: The first column is the comparison between positions, the second column is the comparison between velocities, and the third column is the difference between actual and desired velocities. Each row corresponds to a joint of the iiwa from joint 0 to joint 6.