Robotics and Navigation in Medicine Report

Group 2

1. Introduction

The main aim of the project was to develop a program to control the Franka Emika robotic arm to perform a needle insertion on a skeletal representation of the human torso such that the needle penetrates a provided target in the chest cavity without colliding any bone present. The project involved various aspects of Robotics and computer Vision which were used for controlling the bots dynamics and with the presence of the camera a visual representation of the work environment could be captured. The code was primarily programmed using C++ as the coding language and its integration was done using the ROS framework so that communication with the bot could be established.

The project was divided into two subsystems

I. Kinematics
II. Vision

The team was also divided into groups who would focus on tasks specific to these subsystems and with one member from each group working on the ROS integration. In the following sections a description of the subsystems will be provided.

2. Kinematics

2.1 Direct kinematics

(Noah Pauker)

The direct kinematics were carried out according to the rules of the Denavit-Hartenberg transformation. The input parameters are the angles of the joints and the Denavit-Hartenberg parameters provided by Franka Emika. The output values are the transformation matrices from the base to each individual joint and to the end effector.

2.2 Matlab simulation

(Noah Pauker)

To be able to check our kinematic calculations without much effort, we wrote a Matlab program that plots the panda robot arm in 3D in a quite simple form. With the help of this simulation,

small changes during development can be checked quickly without having to calculate and simulate the entire code in ROS.

The simulation works in such a way that the individual translation vectors of the transformation matrices *T_0 - T_07* are connected with vectors, which are then plotted in a given three-dimensional map. For this the Matlab fiction *plot3 ([X1, X2], [Y1, Y2], [Z1, Z2])* was used, which connects two points in space with a line and plots them.

In connection with a loop, many individual position calculations of the robot arm and in connection with the Matlab commands *clf ('reset');* and *pause (0.001)* movements can also be simulated. This becomes even more interesting when calculating the trajectories.

2.3 Inverse Kinematics

(Noah Pauker)

Inverse Kinematics has the position and orientation of the end effector in the original coordinate system as input values and the angles of each individual joint of the robot arm as output values. The solution for the inverse kinematics was solved according to the geometrical principle, which was presented in tutorial 5. The sequence of the coordinate systems of the Panda robot arm is reversed so that coordinate system 0 is at the tip of the end effector and coordinate system 7 is at the base of the robot arm. First, the joint with the new coordinate designation j1 is rotated so that wrist-offset points in the direction of the basement. This makes the rest of the positioning and orientation a plane problem, which reduces the complexity. In addition, the joint j3 is set to zero to avoid overdetermination. The joints j2 and j4 are then calculated using the law of cosine and the geometry of the robot arm. Finally, the joints j5, j6 and j7 are calculated from the rotation matrix 4R7. The following formula is used to obtain the rotation matrix 4R7.

The positioning worked very well with this solution approach, but there were problems with the orientation of the arm. Unfortunately, we did not find a solution to this problem in the limited time.

2.4 Trajectory Planning

(Noah Pauker)

In trajectory planning, the input value is the position to which the robot arm should move and the time in which it should cover this distance. The output values are matrices filled with the angles at a certain point in time during the movement period, whereby a matrix is created for each joint.

The trajectory is calculated in the first half of the previously determined time, in which the robot arm moves from the current position to the new position, the joints of the robot arm accelerate constantly and from half of the time constantly decelerate. Therefore, the additional conditions of the calculations are that the acceleration and the speed at the start and end are zero and maximum at half the time. The main conditions are the positions at the start and end time. With the help of the main and secondary conditions, two quadratic equations can be calculated which determine the corresponding positions as a function of time. The time interval is

specified by Franka Emika and is 1KHz, depending on the speed, the required time between the start and end point must be specified in advance. If the transferred values exceed the joint space limits given by Franka Emika, the specified start end time is adjusted in the program. If a joint is not moved at all, an array is generated which always has the same angle.

2.5 Trajectory Planning Needle

(Noah Pauker)
In trajectory planning for the needle penetration, the trajectory must follow a straight line between start and end. In solving this problem, we had the idea of dividing the straight route into many individual segments and calculating them like the trajectory planning described above. The only difference should be that for the secondary conditions should apply that the start acceleration and speed of a path section corresponds to the end acceleration and speed of the previous path section. This would result in an even movement that follows the straight line well, depending on how many segments it was divided into. However, we failed in completing this function. So, we used the trajectory planning from the previous section. As a result, the needle follows the straight line similarly well, but the movement is uneven and stops shortly after each path section.

2.6 Path Planning

(Akshay Shetty)

The path planning computes the waypoints of the bots end effector, to move the end effector from an initial point to the desired position within the workspace of the bot.

1. Camera Calibration
2. Model Registration
3. Needle Insertion

Camera Calibration:

A CAD model of the Franka Emika and the checkered board was made based on the provided specification. Coordinates of 15 points were selected such that the camera's area of vision would cover the entire board. Once these 15 points were finalized the points were replicated at 2 more positions along the z axis providing a total of 45 positions.

Model Registration:

The waypoints to capture images of the phantom was done by guiding the bots end effector in the hand guided mode and recording the coordinates that make good waypoints.

Needle Insertion:

Once the coordinate of the target was obtained a vector would be generated along which the final insertion would take place. The vector would be selected such that the points around

the target will not intersect with the vector which in turn would mean that the needle would not encounter any bone. Considering the needle length the end effector would be placed at an offset position. This part of the project could not be implemented.

ROS integration

Nodes were created for individual tasks such as inverse kinematics, Trajectory Planning and Path planning. The inverse kinematics subscribes to the topic that provided the end effector co-ordinates and calculates the individual joint angles of the bot and publishes it.

For Path Planning the coordinates are contained the main class the coordinates are published the target coordinate topic which is subscribed by the inverse kinematics node. Here the joint angles are calculated and published. The published joint angles are subscribed by the trajectory planning node, This node takes the initial joint set and final joint set and calculates the intermediate joint angles based such that the bots motion is within permitted limits and is smooth.

3. Vision

3.1 Intrinsic and Hand-eye Calibration

(John Mockler)

In order to estimate the intrinsic matrix and distortion coefficients of the camera, we must first collect images of the checkerboard from various angles and distances. The robot was directed to move to fifteen poses and pause briefly. A ‘Master’ node constantly checks the pose of the robot and calculates the squared sum of the error of the pose angle to the target pose. If this is lower than an error threshold, the Master node sends a service call to the Calibration node, directing it to capture an image. Once all fifteen target poses have been reached, the Master Node sends a different service call to Calibration, telling it to start the calibration process. The calibration node is subscribed to the raw rgb image output from the Kinect, as well as the pose information from the robot, and stores the most recent data (constantly overwriting the previous information). When it receives the message to capture an image, it stores the most recent image, as well as most recent robot poses into arrays.

Once all calibration images and their related poses have been captured, the Calibration node is then directed to proceed with calibration. For each calibration image, we search for the checkerboard corners using OpenCV’s built in function, input this, as well as an array of known checkerboard locations, into OpenCV’s calibration function to get the camera matrix and distortion.

With the calibrated camera, we then estimate the pose of the checkerboard in each photo using OpenCV solvePNPRansac, which calculates the pose of the checkerboard with regard to the camera.

For each calibration image, we have the estimated transformation of the calibration plane with regard to the camera, and the transformation of the robot end effector with regard to the base frame. With that, we proceed to calculate the hand-eye calibration by solving the following equation for the transformation from end-effector to camera and robot base to calibration simultaneously:

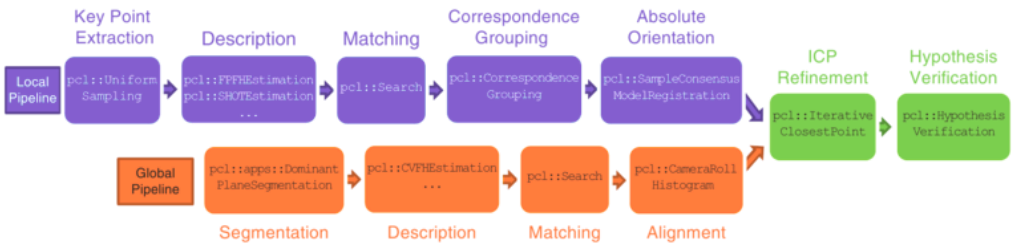
Hend effectorrobot base* Hcameraend effector= Hcalibration planerobot base* Hcameracalibration plane

To do this, we implemented the QR24 method developed by Ernst et. al, particularly because high speed which is important when using on the robot in real time. To evaluate this method, we compared it to the Tsai-Lenz approach implemented by OpenCV, using the pre-recorded marker and robot poses provided to us, and found that they produce similar results.

Finally, using the calculated transformation from end-effect to camera, we periodically calculate the transformation base to camera using the current pose data published by the robot, orthonormalize the result using singular value decomposition, and publish, for use by the Point Cloud node.

3.2 Point Cloud

(Asan Adamanov)
As soon as we completed camera calibration and hand-eye calibration tasks, we could finally find a transformation between the base and the camera. So, the requirements information for working on PointCloud were completed, we could start working on PointCloudNode.
After we collected the necessary information about PCL, we discovered, there exist two different pipelines that we could implement in order to register a 3D Model. However, before we jumped into the registration of the model we had to firstly stitch the clouds. This procedure had to be done because the clouds were taken with the mounted RGB-D camera on the end effector was moving, and we were hoping to generate better a scene cloud if we take a shot of point clouds from different positions and then stitch them into one big scene cloud and could work on it. (it is where we actually failed during testing our code in a lab, and unfortunately, as we did not have enough lab sessions to correct our mistakes here. The Professor said on the last submitting day the mistake could be because we took a shot of clouds from very far location to each other, and respectively the scene cloud could not be generated as we wished). Despite our unsuccessful stitching procedure in the lab, we worked on a saved single point cloud which we had generated with a ROS bag file earlier and tried to register a 3D CAD model to that scene cloud.
As I mentioned before, we found two different ways (called Local Pipeline and Global Pipeline) to register a 3D model. The main difference between the two pipelines as from their name can be recognized, is that the Local Pipeline does process a scene and 3D model clouds locally, what I mean by this, basically, it finds a *keypoints* of each cloud calculate their descriptions, On another side, the Global Pipeline based on the *segmentation* procedure and compute their descriptors. After the descriptors are found, the KD-Tree algorithm is performed and tries to match the scene and 3D Model cloud. After that, we did some post-processing, and finally, a position of the registered model can be calculated. The summarize of the two pipelines is shown below on a picture.



(Example of local and global 3D recognition pipelines in PCL (image from paper by Aitor Aldoma et al.).

3.3 ROS Integration

(Suyash Sonawane)

All individual modules mentioned so far are modelled as individual nodes. Communication between nodes and controlling the flow of program execution is taken care of, by using Subscribers, Publishers and even Services. Subscribers and Publishers proved to be helpful when a constant stream of data was required. For example, the joint states of the robot were constantly streamed and were compared at regular intervals. Here, sync between sending and receiving was not required.

On the other hand, there were sections of the program, where asynchronous communication was of importance. In the Camera Calibration Node, pictures of the checkerboard were to be taken at particular points. It was also important to ensure that the camera does take a picture only when the robot was at a stationary position, and was not in motion. Thus, a Service Client and Server were used to make this possible. Upon reaching a desired position, a Service request is sent to the Calibration Node, which instructs the camera to save the latest image from the input stream. Once saved, a service response is sent back to confirm that the image has been saved successfully.

A similar model has been used to interface the Point Cloud Node with the kinematics part of the system.