Compliant Grasping

Report for the Final Project of
Practical Course RoboCup@Home
in the Institute for Cognitive Systems of Technical University of Munich.

Supervised by Prof. Dr. Gordon Cheng,
Dr.-Ing. Karinne Ramirez Amaro,
M.Sc. Julio Rogelio Guadarrama Olvera,
Dr. Pablo Lanillos Pradas,
M.Sc. Constantin Uhde
Institute for Cognitive Systems

Submitted by Siqi Hu,
Zhenyu Li,
Chensheng Chen,
Chenghao Wang

Submitted in Munich, Feb.12th, 2019

Anhang I

Erklärung

Wir versichern hiermit, dass wir die von uns eingereichten Bericht selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

München, 13.02.2019, Siqi Hu, Chensheng Chen

Ort, Datum, Unterschrift

Contents

1. Introduction

One of the most convenient and advanced ways of moving robot's arm to a desired position and grasp an object and avoid collision with obstacles is using `moveit` motion planning framework. Once the deired goal is sent to the `move_it` node, it will calculate a path that pypasses obstacles and reaches the goal. And by execuing this path, `move_it` will communicate with the physical interface of robot and the robot will be moved.

However, in our final project, we propose to manipulate the robot without using `move_it`. According to some control laws that will implement impedance and admittance control of arm and gripper, some values will be calculated and directly sent to the physical interface of robot. Moreover, skin are used to modify these values sent to robot. By incorporating controllers and skin, complinat grapsing adaptve to the environment is implemented. General structure of the project is as following.

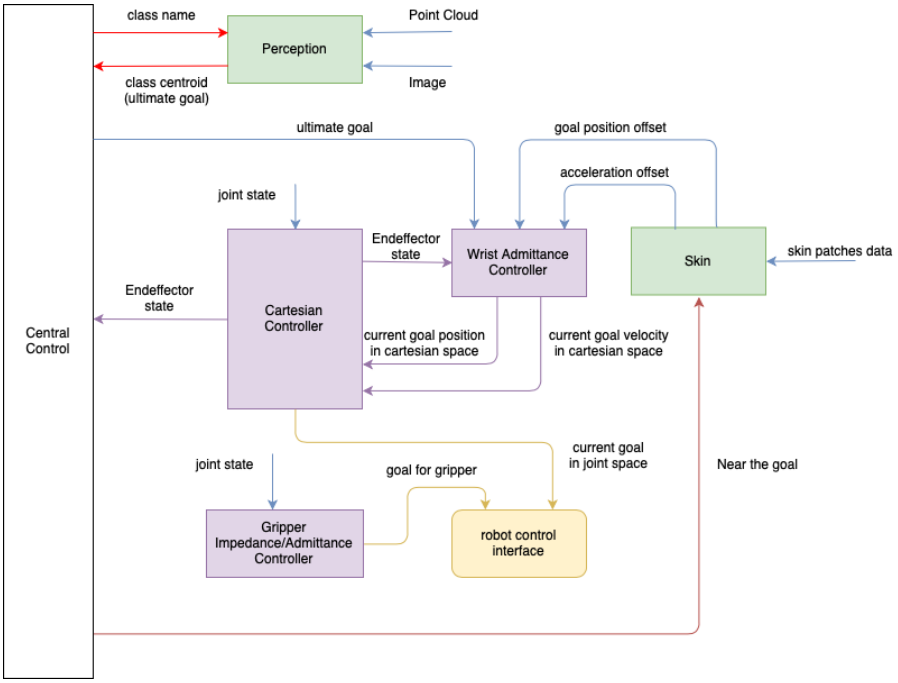


Figure 1 General structure

2. Cartesian Controller

by Siqi Hu

2.1. Abstract

Basic task: Given a goal position in cartesian coordinnate, move the predefined end effector of the arm to this position by directly communicate with the arm controller of `Tigao` instead of using `move-it` planning and execution.
Subtasks: In order to achieve this basic goal, 4 functionalites are to implement one after another. Namely the subsequent 4 subtasks: read `robot description`, forward kinematics for position, inverse kinematics for position, inverse kinematics for velocity.

2.2. Subtask1: Read robot description

For purpose of forward kinematics and inverse kinematics calculation of the arm, it is necessary to know the structure of the `Tiago` arm. To do this, we need to read `robot description` of the robot, which contains the information about its arm. Fortunately, `Tiago` publishes its `robot description` onto the parameter server and thus `robot description` is stored as a parameter in the server. We can simply retrieve this parameter from the parameter server and extract the informatoin about arm.

For the extraction of arm information and other subsequent kinematics calculations, `Kinematics and Dynamics Library (KDL)` are used. By using the function `treeFromString` in the library, a tree that describes the structure of the whole robot is formed. From this tree, a chain class describing the arm is to extract.

We know that arm contains 7 links and its first link and last link are `arm_1_link` and `arm_7_link` respectively. The parent link for `arm_1_link` is `torso_lift_link` and the `arm_tool_link` is one of the child link for `arm_7_link` and lies in the middle between the gripper. Therefore, we can define `arm_tool_link` as the tip of the chain, in other words, the end effector, and define `torso_lift_link` as the root of our chain. In this way, a chain that represents the arm is extracted from the tree. And since the root of our chain is `torso_lift_link`, it is obvious that the calculated position in cartesian coordiante by forward kinematics generates a position in the coordinate frame of `torso_lift_link`. And similarly, if we want to calculate the joint value by giving desired cartesian goal position to inverse kinematics solver, we need to tranform this cartesian goal position to the postion in the coordinate frame of `torso_lift_link`. For example, in perception part, the position of the detected object is generated, but it is in the coordinate frame of camera, in this case, the tranformation from camera coordinate to `torso_lift_link` is required.

On the top of the chain of arm, forward kinematics solver and inverse kinematics solver can be built, which are defined class in `KDL`.

2.3. Subtask2: Forward kinematics for position

The admittance controller of arm on the higher layer needs to know the current cartesian position to do its control. Thus, cartesian controller is supposed to calculate the current cartesian position from current joint value using the previously built forward kinematics solver.

`Tiago` publishes its joint values to the topic `/joint_status`. Thus, joint values of 7 arm joints are easy to acquire. Member function `JntToCart` of the FK solver can convert the joint values to a cartesian position.

This cartesian position is sent to a predefined topic and the controllers on higher layer can subscribe this topic and use this position to do its control.

2.4. Subtask3: Inverse kinematics for position

For admittance control of arm, given a ultimate goal position, a set of current goal positions generated by admittance control will be sent to the cartesian control. Therefore, we need to convert these goal positions into goal joint values.

For this purpose, member function `CartToJnt` of the IK solver can be used, which reads the current joint values and the goal cartesian position in and outputs the goal joint values. Besides, since the IK calculation doesn't take the limits of joint values into account. Manual limitation of joint values are necessary. If the calculated goal joint values are beyond the range, the corresponding goal cartesian position is considered as unreachable and will be discarded.

In addition to that, because of the current goal update scheme of the arm admittance control, current goal positions are only positions that are slightly shifted from the current position, and the accumulation of reaching this set of goals results in reaching our ultimate goal position. Therefore, converted goal joint values should only change a little bit as well comparing to current joint values. If the difference between current joint values and calculated goal joint values is beyond some thereshold, we think this goal would make the robot arm move too violently and would discard this goal. In this way, rapid move of the arm, which may cause damage to the robot, would be avoided.

Joint Name	Minimum Value	Maxmum Value
arm_1_joint	0.02	2.73
arm_2_joint	-1.55	1.07
arm_3_joint	-3.51	1.55
arm_4_joint	-0.37	2.34
arm_5_joint	-2.07	2.07
arm_6_joint	-1.41	1.41
arm_7_joint	-2.07	2.07

Table 1 Range of joint values

2.5. Subtask4: Inverse kinematics for velocity

Arm controller in `Tiago` subscribes the topic `/arm_controller/command` and executes the corresponding command to move the arm. Important data that need to be filled into the command message are joint names, joint values, velocities of joints when they reach their goal, time that joints would use to reach the goal.

In order to simulate a uniformly accelerated motion in the current minor movement towards the current goal that is slightly shifted from current position, joints need to have a velocity, that is calculated and published by the admitance controller, when joints reach the current goal position. Since the published velocity is in cartesian coordinate, a inverse kinematics for velocity is implemented to generate the goal velocities of joints. Then, besides filling in the joint name, joint value, time from start, goal velocities of joints are filled into the command message as well.

For implementation of inverse kinematics for velocity though, there is problem in the IK velocity solver of the `KDL` library. Execution of member method `CartToJnt` in IK velocity solver always leads to error. Thus, we abandon this method and do IK for velocity manually.

Fortunately, the previous IK solver for position works properly and generates a Jacobian matrix for us. We can use singular value decomposition of the Jacobian matrix to calculate its pseudo inverse matrix. Pre multipy the goal cartesian velocity by the pseudo inverse matrix, the result is the goal joint velocity.

To this end, we fill in the corresponding joint values and joint velocities into the command message and publish it to the arm controller. The arm then would move accordingly.

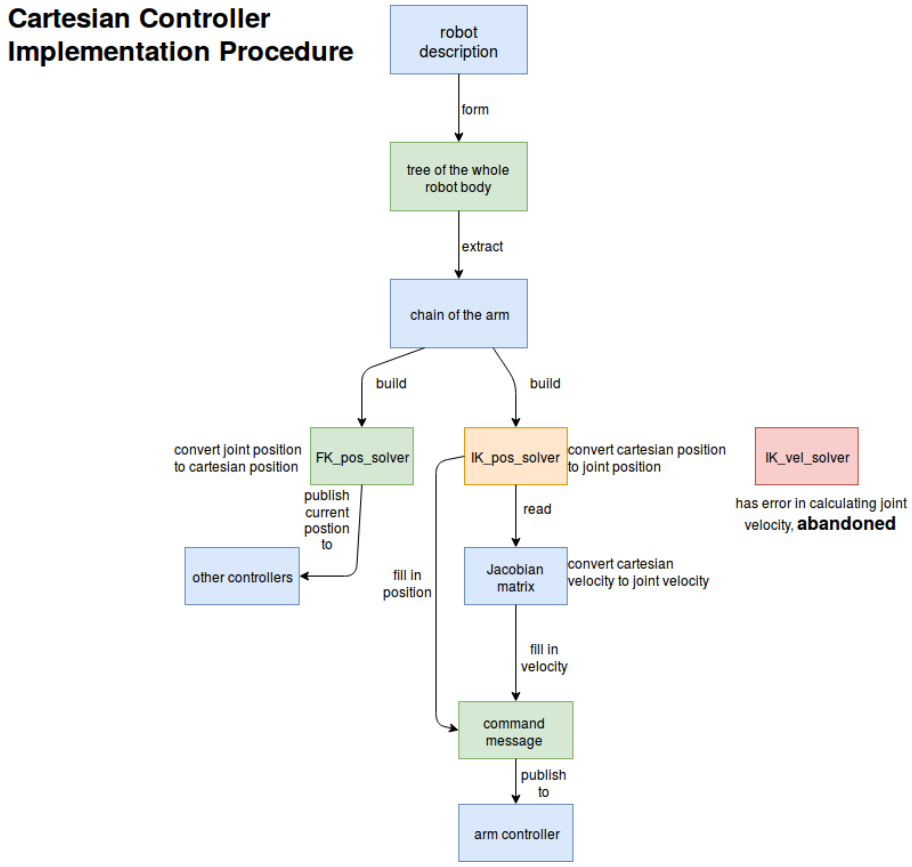


Figure 2 Cartesian controller implementation procedure

3. Perception

By Zhenyu Li

3.1. Pipeline

In the final project, we implemented a new pipeline, to make it more reliable. Figure 3 shows the structure of the system.

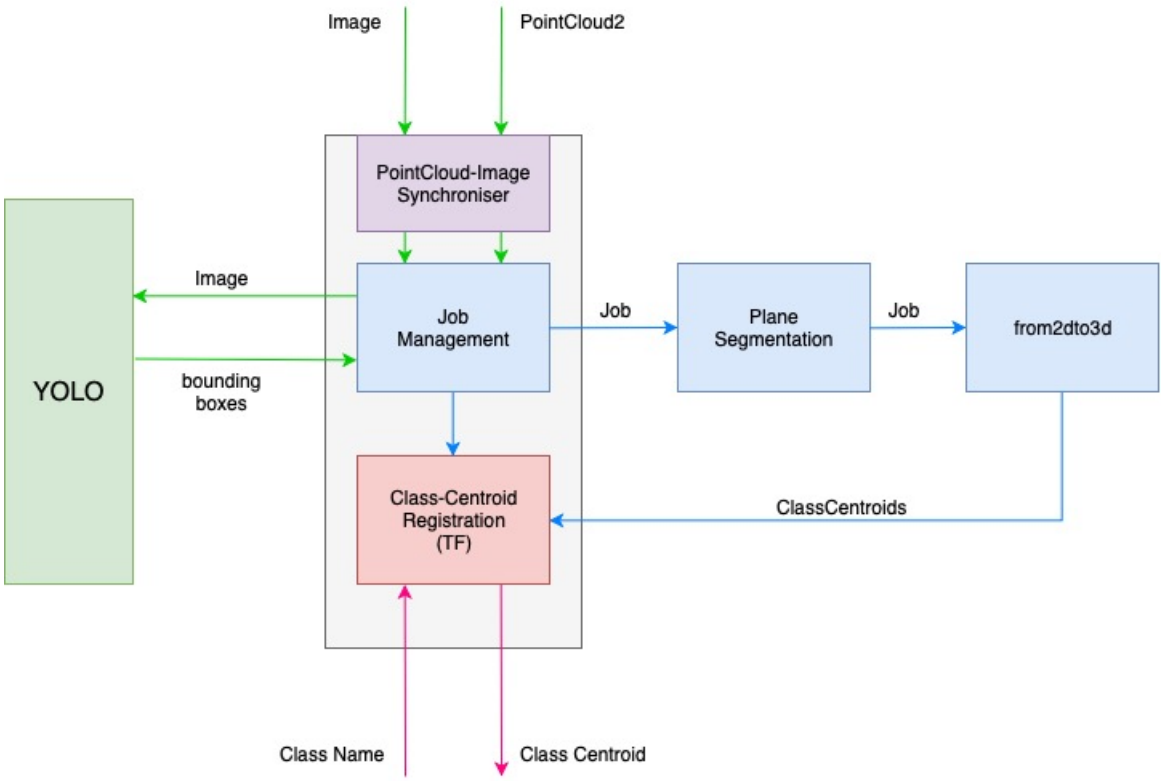


Figure 3 Pipeline for perception system

There are 5 main functional blocks, following sections will explain what is going on in each block and how they works.

3.2. Synchroniser

The calculation of point cloud is rather time consuming, in most cases, the point cloud is updated with much time delay. If the scene is changing or the camera is moving, we must take the timing problem into consideration, namely, point cloud and image must be synchronised in terms of time stamp.
In this block, an image buffer is implemented to buffer the incoming images, when a new

point cloud message is received, the synchroniser will firstly compare the time stamp between buffered images and point cloud, the the image with lowest time difference is picked as a matched image for that point cloud.

3.3. Job Management

The matched image-point cloud pair is passed to this block, the matched image is then sent to YOLO, after the bounding boxes is returned, a new Job is generated in the block. The Job message contains the point cloud and the bounding boxes, it will be processed in 2 stages.

3.4. Plane Segmentation

This block was written in one node to perform plane segmentation. Some information about the range of the original point cloud is added in the job message, because they are only available before segmentation, and they are important for the next stage. The point cloud in the job message is then replaced by the point cloud without large plane.

3.5. from2dto3d

This block was written in one node to calculate the centroid of each class in the bounding boxes. Since the calibration matrix or camera intrinsic matrix is unknown, we have to project the points in the point cloud back to picture plane, to determine wether a point is within a bounding box or not. Note that the bounding box is on the pixel plane, we need to perform another projection from the pixel plane to the picture plane. Traversing all the points, and note down which points are belong to which bounding boxes. The Centroids for each bounding boxes is calculated with a pcl built-in function, and then it is sent back to the main node.

Figure 4 Original point cloud

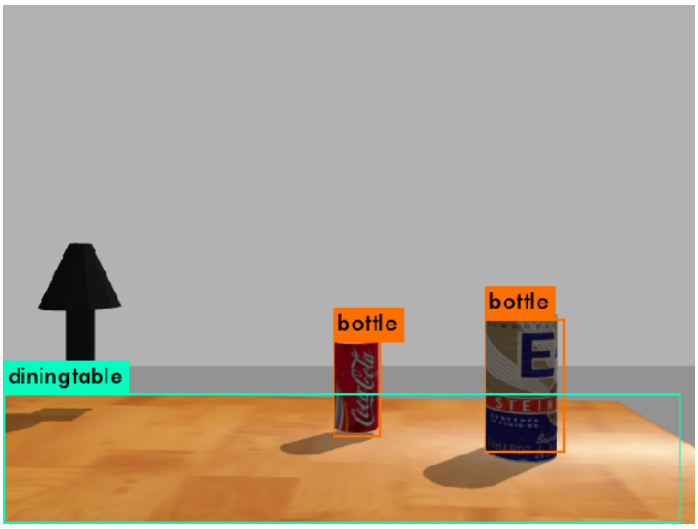


Figure 5 Bounding boxes in YOLO

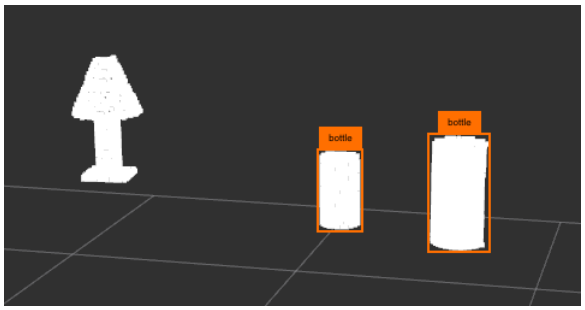


Figure 6 Point cloud after plane segmentation with projected bounding box

3.6. Class-Centroid Registration

Once the result from from2dto3d is received, the block performs a TF transformation from camera link to a fixed link to the centroids, the transformed centroids are then stored in a local vector.

4. Admittance Controller using wrist force-torque sensor

By Zhenyu Li

4.1. Abstract

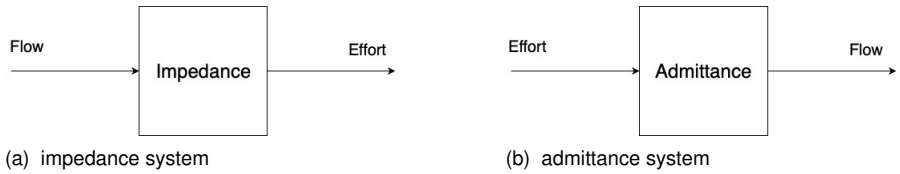
Compliance control for rigid-body manipulators is always an important topic in the field of robotics, since a specific contact force may required in many scenarios like medical robots, soldering robot etc. The robot it self should also be protected against damage caused by collision with environment.
Think about a robot is grasping a small object like an apple on a table, centroid of the object is very close to the hard surface of the table, since the data given by visual system are always found to be not sufficient accurate, there exists a huge danger, if some parts of the robot hit the table, and the impact force is so great, that the gripper or arm of the robot may broken. If robot's action could be less "rigid", which is called compliant impedance behavior, the impact would be harmless to the robot. Impedance behavior can be imposed by admittance control via additional measurement of external forces using a force-torque sensor, which is available in Tia-Go. To avoid the accident mentioned above, an admittance controller is introduced to our project.

4.2. Fundamental of Admittance Control

Physical power could be represented by the product of flow and effort, eg. in electrical system, the effort is the voltage, while the flow is the current, it's true that $P = U \cdot I$. Similar in mechanical system, however in this case the effort is the force/torque, and the flow is the velocity/angular velocity, and the power is given by $P = \boldsymbol{F} \cdot \boldsymbol{v}$ or $P = \tau \cdot \omega$.
The terms Impedance and Admittance originate from electrical system, to determine wether a system performs a impedance or admittance behavior, we need to tell, what is the cause and what is the result. eg. we apply voltage on a resistance, and we measure the current as a result, the transfer function of this system is $1/R$, which is admittance. Figure 7

Impedance behavior could be described as the following spring-damper model.

Position of the mass M is denoted by x. The control input is given by F_{env}. In this section we analyse only one dimension. With the spring constant K and damping constant D, original



(a) impedance system (b) admittance system

Figure 7 Impedance system and admittance system

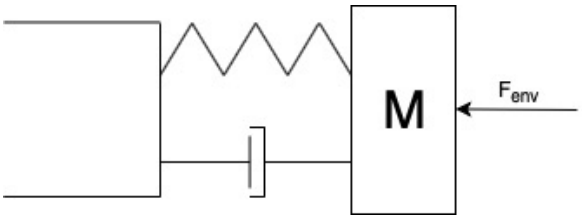


Figure 8 Spring-Damper Model

length of the spring is x_r. The dynamics of this system is therefore given by the following relation $\Im(x, \dot{x}, F_{env})$:

$$M\ddot{x} + D\dot{x} + K(x - x_r) = -F_{env} \tag{4.1}$$

4.3. Implementation

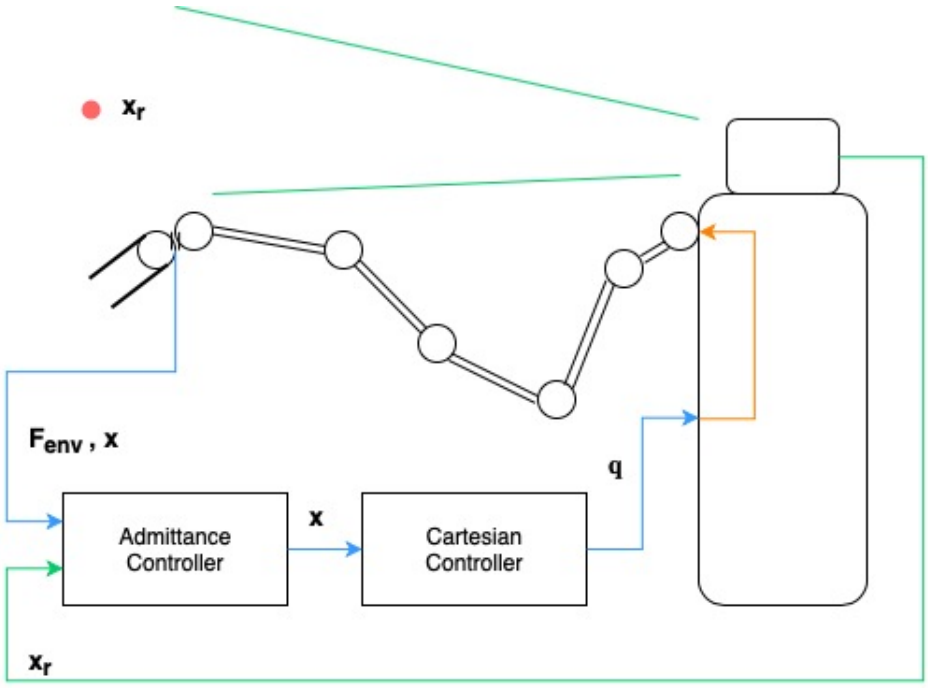


Figure 9 Cartesian-Admittance control using wrist force-torque sensor

Since the position controller interface is provided in Tiago, it is always equipped with a force-torque sensor in the wrist, one possible solution is shown in Figure 9 .
We rewrite the relation Equation (4.2) in 3 dimensional cartesian space with unified positive direction,

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{D}\dot{\boldsymbol{x}} + \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{x_r}) = \boldsymbol{F_{env}} \tag{4.2}$$

after some simple reformation we have,

$$\ddot{\boldsymbol{x}} = \frac{1}{\boldsymbol{M}}\left[-\boldsymbol{D}\dot{\boldsymbol{x}} + \boldsymbol{K}(\boldsymbol{x_r} - \boldsymbol{x}) + \boldsymbol{F_{env}}\right] \tag{4.3}$$

we can easily calculate velocity and cartesian position for each time interval by using numerical integration,

$$\begin{aligned} \dot{\boldsymbol{x}} &\coloneqq \dot{\boldsymbol{x}} + \ddot{\boldsymbol{x}}dt \\ \boldsymbol{x} &\coloneqq \boldsymbol{x} + \dot{\boldsymbol{x}}dt \end{aligned} \tag{4.4}$$

the cartesian position $\boldsymbol{x}$ is then sent to cartesian controller, where the cartesian position is transformed into joint space and sent to the robot control interface.
Since the part in front of the wrist force-torque sensor has its own weight, and the sensor itself accumulates some inner stress, which should not be neglected. These forces must be compensated before we use the reading as F_{env}, otherwise the robot may not able to reach a certain position accurately.
Inner stress and gravity should be compensated separately in two different coordinate, because gravity always results perpendicularly to the sea level, and the inner stress is a constant relative to the coordinate of the sensor.

5. Gripper Control: Impedance And Admittance Control

By Chenghao Wang and Chensheng Chen

5.1. Control Law

The control of the gripper is chosen as impedance and admittance control, which can provide very good performance for interaction between gripper and environment. As we already discussed in chapter 4, both impedance control and admittance control can be described with following formula:

$$M\Delta\ddot{x} + B\Delta\dot{x} + K\Delta x = F_{ext} \tag{5.1}$$

M, B, K are desired inertia, damping and stiffness, they are defined by us to achieve the deigned dynamics. This kind of mechanism can be controlled with force in impedance control and acceleration in admittance control. $\Delta\ddot{x}, \Delta\dot{x}$ and Δx are the difference between current states $\ddot{x}_c, \dot{x}_c, x_c$(from the sensor) and desired states $\ddot{x}_d, \dot{x}_d, x_d$. The gripper has two target motions in this task: grip and release. The desired states of these two motions are defined as follow:

$$\ddot{x}_d = 0, \dot{x} = 0, x_d = \begin{cases} 0, & grip \\ 0.04, & release \end{cases} \tag{5.2}$$

Combining (5.1) and (5.2), the control function can be written as:

$$M\ddot{x}_c + B\dot{x}_c + K(x_c - x_d) = F_{ext} \tag{5.3}$$

The dynamic of the gripper should be controlled as (5.3).

5.2. Practical Problem

Due to some limitations of the robot, the control law above can't be implemented directly and should be modified to adjust to the robot. The first problem is the lack of hardware interfaces for force and acceleration. As a result, the control of force and acceleration should be converted to velocity and position:

$$\dot{x}(t + dt) = \dot{x}(t) + \ddot{x}(t + dt) \cdot dt \tag{5.4}$$

$$x(t + dt) = x(t) + \dot{x}(t + dt) \cdot dt \tag{5.5}$$

$(5.4), (5.5)$ calculate an approximation of x and $\dot{x}$ with $\ddot{x}$ from (5.1). Then the desired velocity and position can be published to PID controller to control the gripper.
The second problem is the limitation of the actuator. In each iteration, the velocity changes $\ddot{x}(t+dt) \cdot dt$. It is so small that the actuator can't deal with it and the states will stay the same. This problem can be solved with a virtual velocity defined by ourself. Unlike the velocity from the sensor, the virtual velocity can accumulate itself regardless of the actual change of the gripper, so that the gripper can always move as desired.

5.3. Control Structure

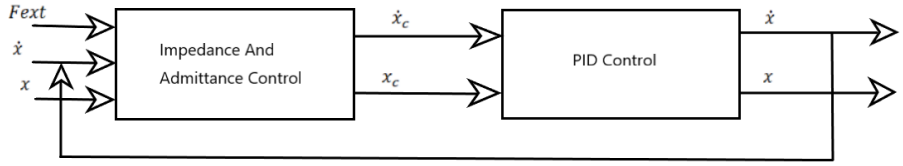


Figure 10 Control Structure

Figure 10 shows the structure of the impedance and admittance control. In each iteration, the controller get the force, velocity and position from the sensor and calculate the desired acceleration with equation (5.3). Then it is converted to velocity and position with equations $(5.4), (5.5)$ and send it to gripper position controller to implement the desired states.

5.4. Stability Analysis

The standard impedance and admittance control is stable but the modified version can be sometimes unstable. The system defined by $(5.4), (5.5)$ is a simple discrete time system. We can analyze the stability conditions and then set the parameter M, B, K to form a stable system.
Equation (5.1) can be written as:

$$\ddot{x}(k+1) = (F_{ext} - B\dot{x}(k) - Kx(k)) \cdot \frac{1}{M} \tag{5.6}$$

Take it into equations $(5.4), (5.5)$, we get:

$$\dot{x}(k+1) = \dot{x}(k) + \ddot{x}(k+1)dt = \left(1 - \frac{B}{M}dt\right) x(k) - \frac{K}{M}dt \cdot x(k) + \frac{F_{ext}}{M}dt \tag{5.7}$$

$$x(k+1) = x(k) + \dot{x}(k+1)dt = \left(1 - \frac{B}{M}dt\right) dt\dot{x}(k) + \left(1 - \frac{K}{M}dt^2\right) x(k) + \frac{F_{ext}}{M}dt^2 \tag{5.8}$$

Transform them to matrix form, we get the standard discrete time system:

$$\begin{bmatrix} x(k+1) \\ \dot{x}(k+1) \end{bmatrix} = \begin{bmatrix} 1 - \frac{K}{M}dt^2 & \left(1 - \frac{B}{M}dt\right) dt \\ -\frac{K}{M}dt & 1 - \frac{B}{M}dt \end{bmatrix} \begin{bmatrix} x(k) \\ \dot{x}(k) \end{bmatrix} + \begin{bmatrix} \frac{dt^2}{M} \\ \frac{dt}{M} \end{bmatrix} F_{ext} \tag{5.9}$$

Calculate the eigenvalue λ_1, λ_2 of the system matrix and use the stability condition:

$$|\lambda_1| < 1, |\lambda_2| < 1$$

Finally, we get the stability condition:

$$\begin{cases} 1 - \frac{B}{M}dt \\ \left|\frac{1}{2}\left(\frac{K}{M}dt^2 - 2 + \frac{B}{M}dt\right)\right| < 1 \end{cases} \tag{5.10}$$

The selection of parameters are all based on condition (5.10)

5.5. Extension

According to equation (5.1), the gripper will perform a force to the object when it can not get to the target position. If the object is hard, the force can be big and may damage the gripper. As a result, two modes with different parameters are provided to avoid this situation.

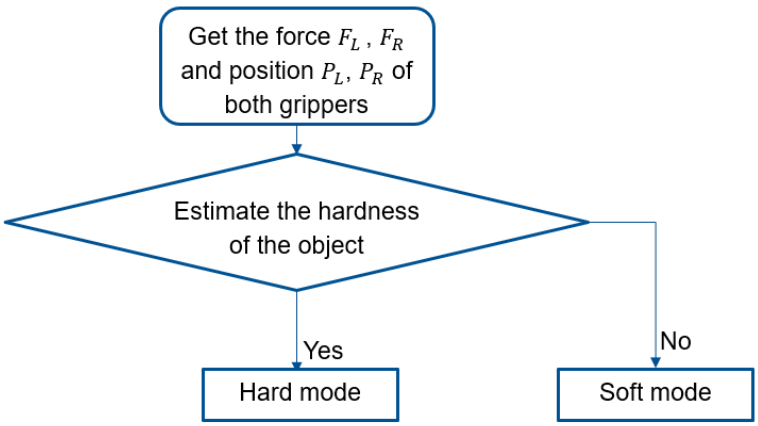


Figure 11 Hard and Soft Modes

Figure 11 shows the process for mode recognition. First, the force and position are obtained with sensor. Then the hardness of the object can be estimated. If the hardness is greater than threshold, the hard mode will be chosen and the force will be reduced. Otherwise, the soft mode will be chosen.

6. Movement Correction using skin

By Chensheng Chen and Chenghao Wang

6.1. Usage of Skin

There are ten patches on the arm of tiago. Patch1 and Patch2 respectively have only one cell. Patch4 and Patch6 respectively have two cells. Patch3 and Patch5 respectively have four cells. Patch7 and Patch9 have respectively fourteen cells. Patch8 and Patch10 have respectively six cells.

For every patch, we can use the topic "/tiago/patchn"(n=1∼10) get the four values of parameters "Force, Proximity, Acceleration and Temperature" of each cell. In every patch, when the obstacle move closer to skin, we can get the the greatest value of proximity. The greatest value can be used to calculate the correctional acceleration Δa. With Δa we can calculate the correctional velocity Δv and the correctional position Δx. Then, the arm can be controlled moving away from the obstacle.

6.2. Flow Chart of Movement Correction

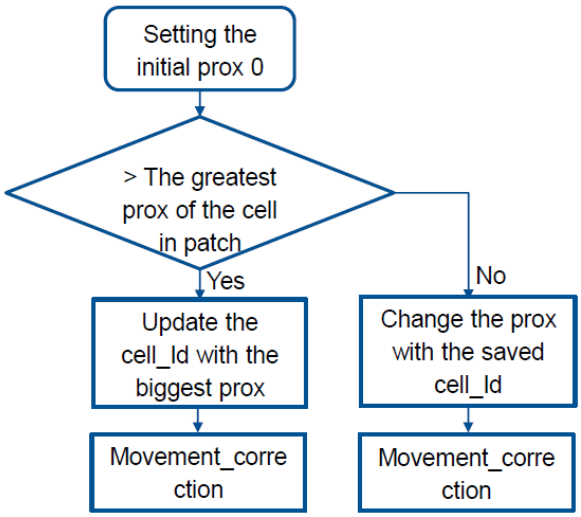


Figure 12 Flow chart of getting the maximum proximity

When the obstacle moves close to one patch of the skin, we can get the greastest value of proximity of the cell. At the same time, we can also keep the Cell_ID. Then, the arm do movement correction. When the arm move away from the obstacle. The recorded value of proximity will also decrease.

6.3. Control Strategy

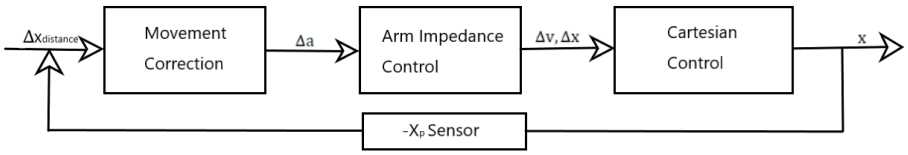


Figure 13 Closed loop control block diagram of movement correction

By the skin, we can get the proximity between skin and obstacle. Using the the proximity calculate the correctional acceleration Δa. The correctional accelerational Δa is sent to the node “Arm Impedance Control”. In this node, the correctional acceleration Δa is used to calculate out the correctional velocity Δv and the correctional position Δx. When the node "Cartesian Control" get the correctional velocity Δv and the correctional position Δx, the arm will move away from the obstalce until the Δx equals to 0.

6.4. Cost Function

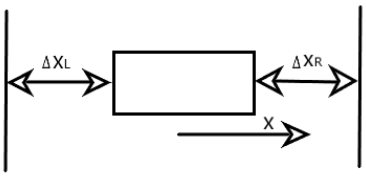


Figure 14 Illustration for the situation of avoiding obstacles

ΔX_L and ΔX_R mean respectively the distance between obstacle and right patch and the distance between obstacle and left patch.
Here, assume proximity $= \frac{1}{\Delta x}$, $F = (\frac{1}{\Delta X_L})^2 + (\frac{1}{\Delta X_R})^2$
Through the derivation, we can get $\frac{\partial F}{\partial x} = -\frac{1}{2}\Delta x_L^{-\frac{3}{2}} * \frac{\partial \Delta x_L}{\partial x} - \frac{1}{2}\Delta x_R^{-\frac{3}{2}} * \frac{\partial \Delta x_R}{\partial x}$

$$= \Delta x_L^{-\frac{3}{2}} - \Delta x_R^{-\frac{3}{2}}$$

So, we can get $\Delta \dot{x} = \Delta x_L^{-\frac{3}{2}} - \Delta x_R^{-\frac{3}{2}} \Rightarrow prox_L^2 - prox_R^2 \Rightarrow (max\ prox_L)^2 - (max\ prox_R)^2$

$$\Rightarrow \Delta a = w * (prox_L^2 - prox_R^2)$$

Here, when the obstacle very close to left patch, Δx_L will be very small, reciprocal of distance Δx_L will be very large, namely $prox_L^2$ will be very large. Δa will be positive and the arm will move right. So, the diatance between obstacle and left patch will be larger. There will be the same situation, when the obstacle close to the right patch, the arm will move left.

6.5. Optimizations

6.5.1. Avoiding sharp acceleration

When the obstacle is very close to the patch. The $prox_L^2$ or $prox_R^2$ will be very large. In this time, the calculated Δa will be very large. The calculated Δv and Δx will be also very large. The arm will in a very short time move very quickly. For avoiding this situation, we add the limit of acceleration a. When the acceleration a is too large, a reasonable acceleration a_{max} will be used instead of the large acceleration a. With this, the movement of the arm will be very smooth.

6.5.2. Eliminating noise of sensor

When no obstacle close to the patch, there are also a little noise of proximity, which come from the sensor. In this case, the arm will always shake. To solve this problem, we just set the mimimum threshold of Δa_{min}. When the calcuiated a smaller than Δa_{min}. This Δa will be not given to control the movement of the arm. It will be thrown away.

6.6. Position Correction

There are four cases before the gripper grip the object.

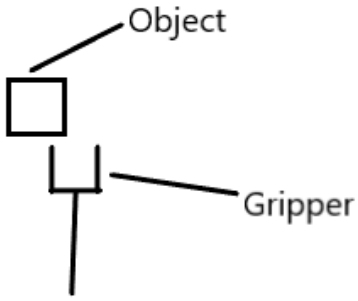


Figure 15 Imperfect grasping situation 1

The first is that, the goal point of gripper is a little right. In this case, the left front patch1 will detect the object. The patch will send the position correction Δx (Δx < 0) to the arm. Then the arm will move a little left. The arm will adjust until there is no object in front of the patch.

The second is that, the goal point of gripper is a little left. In this case, the right front patch2 will detect the object. The patch will send the position correction Δx (Δx > 0) to the arm. Then the arm will move a little right. The arm will adjust its position until there is no object in front of the patch.

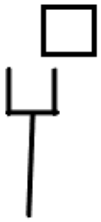


Figure 16 Imperfect grasping situation 2

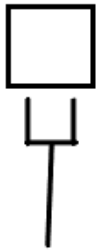


Figure 17 Imperfect grasping situation 3

The third is that, the object is too big. So the gripper can't grip. In this case, the two front patches will both detect the object. The gripper will not move. The information "Can't grip" will be shown in the terminal.

Figure 18 Perfect grasping situation

In the last case, the object is just in the front middle of the arm. The front left and front right patch both will not detect the object. So, in this case, the position of gripper don't need to adjust any more. The arm just need to move a little forwards and grip the object.

Practical problem

In practice, the performance of the position correction is not so good. Because the distance of position correction is too short comparing with the distance of arm's motion. At the same time, if we want to realize the postion correction, the gripper must move very close to the object. This is not so easy to be controlled.

6.7. Temperature

The temperture can be also got from the patches. When the robot grip the object, the two inside patch3 and patch5 will directly touch it. So we can get the temperature of the object. When the temperature of the object is above 50 degrees, the robot will speak "Too hot". When the temperature of the object is below 10 degrees, the robot will speak "Too cold". If the temperature of the object is between 10 degrees and 50 degrees, this is the normal temperatue, so the robot will not speak anything.

Practical problem

The plastic layer of rhe skin is too thick. So when the temperature of the object is too high or too low, after the gripping, the robot will wait for a long time, then it will speak "Too hot or Too cold".