20170699 최진성 Lab Report

Lab 6. System Integration

1. Purpose

Lab 6의 목적은 lab 1부터 lab 5까지의 내용들을 모두 활용하여 하나의 system으로 integrate 하는데 있다. 본 lab에서는 robot manipulation system과 teleoperation을 구축하는데 그 구성요소는 다음과 같다.

- Composed of two computers: PC(user interface), BBB(interacting with the environment)
- Vision: During execution, task scene is obtained by BBB(cam, comm.)
- UI, Teleoperation: The user commands where to pick and place(UDP, CLI interface)
- Teaching, Trajectory generation, Control: BBB can automatically execute predefined motions (interpolation, UDP, state machine, control)

2. Experiment Sequence

본 lab은 4가지 problem으로 구성되어 있다. 이를 통해 predefined sequence of tasks를 automatic하게 수행하게 하는 것이 목적이다. 그러기 위해서는 이전 lab들에서 수행한 direct teaching, task-space control, teleoperation 등을 이용하고 finite state machine(FSM)의 concept을 활용하여야 한다. 아래는 구현하게 될 4가지 problem이다.

(1) Problem 6A. Modified Direct Teaching

Lab 5에서는 direct teaching을 이용하여 manipulator가 3개의 waypoints를 반복적으로 따라가게 하였다. Lab 6에서는 조금 변형되어 robot이 predefine된 2개의 waypoints를 따라가고 최종 position을 유지하도록 한다. 이를 위해 'mode'가 정의된다. Mode가 0일 때는 robot은 두개의 waypoints, initial joint angles와 'waypoints.csv' 파일 첫번째 row에 지정된 user-defined waypoints, 이를 따라간다. Mode change를 위해 error가 정의되는데 error는 'currentQ – user_defined_waypoint'로 설정한다. Error값이 작을 경우 task를 성공했다 판단하고 mode가 0에서 1로 변화한다. Mode가 1일 경우 waypoints는 current position과 두번째 row의 user-defined waypoint로 구성된다. Final waypoint에 도달했을 때 robot은 해당 position을 유지한다.

(2) Problem 6B. Teleoperation using UDP communication

UDP communication을 이용하여 teleoperation을 구현하는 task이다. Lab 2에서 배운 GPIO control과 lab 5에서 배운 task space control을 UDP communication을 통해 구현한다. PC shell에서의 keyboard input을 통해 end-effector인 magnet을 켰다 끄고 task space control 동작을 제어한다.

(3) Problem 6C. Finite State Machine

Finite state machine(FSM)의 concept을 활용하여 이전에 구축한 요소들을 하나의 sequence로 구현한다. 구현하게 될 finite state machine은 아래와 같다.

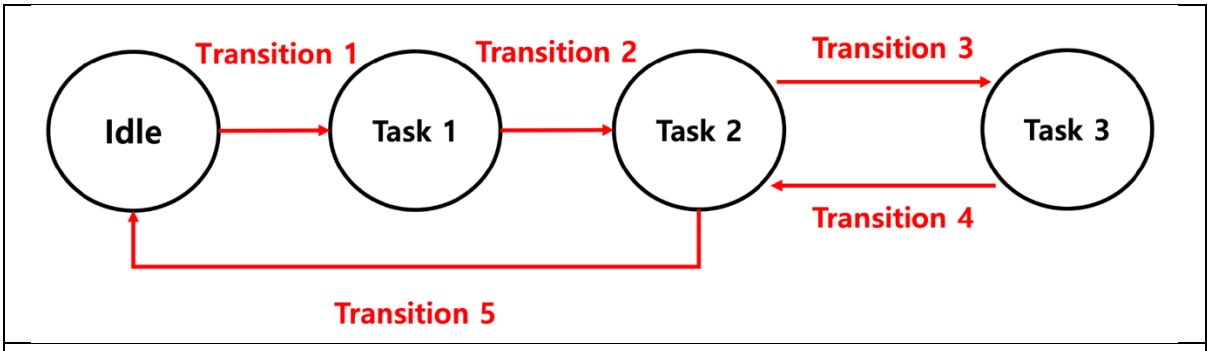


그림 1. Lab 6의 finite state machine

Tasks와 transition condition은 아래와 같다.

States:
- Idle: The robot maintains its current position.
- Task 1: Using a direct teaching method, move the end-effector of the robot to near the target object.
- Task 2: Using the keyboard input from Ubuntu PC, control the end-effector position of the robot and turn on/off the solenoid magnet to pick or place a target object. This process must use visual feedback through a camera.
- Task 3: Using a direct teaching method, move the end-effector to the position where the object is to be placed.

Transition:
- Transition 1: When the keyboard input is 't'.
- Transition 2: The error norm between q and last way point is less than a certain threshold.
- Transition 3: When the keyboard input is 'e'. (When object pick is success, the user gives the input 'e')
- Transition 4: The error norm between q and last way point is less than a certain threshold.
- Transition 5: When the keyboard input is 'x'. (When object place is success, the user gives the input 'x')

(4) Problem 6D. Choose your own problem

Robot Manipulator와 관련하여 지금까지의 배운 내용 외에 수행할 수 있는 task를 설정하여 수행하였다. 내가 설정한 Problem은 다음과 같다.

: Robot Manipultor가 end-effector를 이용해 물건을 옮기거나 외부에서 인위적인 힘이 가해질 때 motor가 견딜 수 있는 부하보다 큰 힘이 주어질 때 알림(led 점등, 알람 등)이 발생하여 안전 예방을 할 수 있는 system 구축

3. Experiment Results

Problem 6A. Modified Direct Treaching

가장 먼저 개발 환경을 구축하였다. Beaglebone과 manipulator, magnet을 lab 2와 lab5에서 한 것처럼 연결하였다. NFS를 활성화 하였으며 Beaglebone을 dynamixel U2D2에 연결하고 U2D2에 power를 공급하였다. Latency timer를 16ms에서 1ms로 변경하였다. 다음 lab 5에서 진행한 것과 같이 waypoints.csv 파일에 두개의 waypoints를 저장하였다. 다음으로 EE405_SystemIntegration directory의 CMakeLists.txt파일을 problem 6A를 위한 executable build를 위해 수정하였다.

```
##for Problem1
add_library(EE405_SI_P1 STATIC include/Controller/ArticulatedSystem.cpp
                               include/Common/Kinematics.cpp
                               include/Common/gpio_control.cpp
                               include/FileIO/MatrixFileIO.cpp
                               include/Common/DirectTeaching.cpp
                               )
add_executable(modified_direct_teaching src/modified_direct_teaching.cpp)
target_link_libraries(modified_direct_teaching Threads::Threads EE405_SI_P1 dxl_sbc_cpp)
```

그림 2. CMakeLists.txt파일을 problem 6A를 위해 수정한 모습

이후 trajectory generation을 위해 EE405_SystemIntegration/include/Common/ directory의 DirectTeaching.cpp 파일을 완성하였다. Lab 5와의 차이점은 'DirectTeaching::TrajectoryGeneration'의 input이 'task_waypoints'이며 class member variables를 사용하지 않고 function input으로 사용할 것이라는 것이다. 이 function의 output은 'task-waypoints'의 각 row인 waypoint를 따라가는 trajectory이다. 해당 function을 구현한 모습은 아래와 같다.

```
// The input argument is the task_waypoints matrix where each row represents a waypoint for direct teaching in joint space
// The output argument is the targetQ vector which is the desired joint angle at the current time step
// The targetQ vector is calculated by using the 1st order polynomial interpolation between the start point and the end point
// The difference with lab5 is that if the end point is the last waypoint, the targetQ vector is set to the desired q.
Eigen::VectorXd DirectTeaching::TrajectoryGeneration(const Eigen::MatrixXd &task_waypoints)
{
    Eigen::VectorXd targetQ(dof); targetQ.setZero();
    int num_task_waypoints = task_waypoints.rows();

    for(int j=0; j<dof; j++){
        targetQ(j)= (task_waypoints(EndPointIndex, j) - task_waypoints(StartPointIndex, j))/NumNodes*NodeIndex + task_waypoints(StartPointIndex, j); /* Implement here. targetQ(j) is the result of the first order polynomial interpolation
        between task_waypoints(StartPointIndex, j) and task_waypoints(EndPointIndex, j) */
    }

    // If the node index is greater than or equal to the number of nodes,
    if(NodeIndex>=NumNodes)
    {
        if(EndPointIndex>=num_task_waypoints-1) // If the end point is the last waypoint
        {
            targetQ = task_waypoints.row(EndPointIndex); /* Implement here. Set the targetQ as the final waypoint */
        }
        else // If the end point is not the last waypoint
        {
            //update the 'StartPointIndex' and the 'EndPointIndex' and reset the 'NodeIndex'
            StartPointIndex++; /* Implement here*/
            EndPointIndex++; /* Implement here*/
            NodeIndex = 0; /* Implement here*/
        }
    }
    else
    {
        NodeIndex++;
    }

    return targetQ;
}
```

그림 3. DirectTeaching::TrajectoryGeneration function의 모습

위에서 보면 end point가 last waypoint일 경우 targetQ를 final waypoint로 지정하는 모습을 볼 수 있다. 즉, robot이 final waypoint를 maintain할 것이라는 것을 확인할 수 있다. 최종적인 DirectTeaching.cpp 코드는 아래와 같다.

```
/**
 * @file DirectTeaching.cpp
 * @author Jinyeong Jeong (jinyeong.jeong@kaist.ac.kr)
 * @brief
 * @version 0.1
 * @date 2023-04-04
 *
 * @copyright Copyright (c) 2023
 *
 */
#include "DirectTeaching.hpp"

DirectTeaching::DirectTeaching(int dof_args)
{
    NumWaypoints=3; // the number of waypoints
    dof=dof_args; // DOF of the manipulator
    stack_waypoints.resize(NumWaypoints, dof); // Each row in the stack_waypoints matrix
represents a waypoint for direct teaching

    NumNodes=400; // the number of nodes
    NodeIndex=0; // node index. The range of NodeIndex is 0 ~ (NumNodes-1)
    StartPointIndex=0; // Waypoint index of start point in trajectory generation. The range
of StartPointIndex is 0 ~ (NumWaypoints-1)
    EndPointIndex=1; // Waypoint index of end point in trajectory generation. The range of
EndPointIndex is 0 ~ (Numwaypoints-1)

}
```

```
// Define stack_waypoints where each row represents a waypoint for direct teaching in joint
space
// The first waypoint for direct teaching is the initial joint angle when the program runs
void DirectTeaching::initialization(const Eigen::VectorXd &current_q)
{
    // Save two user-defined waypoints to the stack_waypoints using the "waypoints.csv" file
    // Note that the first waypoint for direct teaching is the initial joint angle when the
program runs
    stack_waypoints     <<current_q(0),     current_q(1),     current_q(2),     current_q(3),
MatrixFileIO::openData("waypoints.csv");

    std::cout << "Each row represents a waypoint for direct teaching in joint space: \n" <<
stack_waypoints<< std::endl;
}

// Clear the variables for direct teaching
void DirectTeaching::clear()
{
    NodeIndex=0;
    StartPointIndex=0;
    EndPointIndex=1;
}

// The input argument is the task_waypoints matrix where each row represents a waypoint for
direct teaching in joint space
// The output argument is the targetQ vector which is the desired joint angle at the current
time step
// The targetQ vector is calculated by using the 1st order polynomial interpolation between
the start point and the end point
// The difference with lab5 is that if the end point is the last waypoint, the targetQ vector
is set to the desired q.
Eigen::VectorXd DirectTeaching::TrajectoryGeneration(const Eigen::MatrixXd &task_waypoints)
{
    Eigen::VectorXd targetQ(dof); targetQ.setZero();
    int num_task_waypoints = task_waypoints.rows();

    for(int j=0; j<dof; j++){
        targetQ(j)=   (task_waypoints(EndPointIndex,   j)   -   task_waypoints(StartPointIndex,
j))/NumNodes*NodeIndex + task_waypoints(StartPointIndex, j); /* Implement here. targetQ(j)
is the result of the first order polynomial interpolation
        between task_waypoints(StartPointIndex, j) and task_waypoints(EndPointIndex, j) */
    }

    // If the node index is greater than or equal to the number of nodes,
    if(NodeIndex>=NumNodes)
    {
        if(EndPointIndex>=num_task_waypoints-1) // If the end point is the last waypoint
        {
            targetQ = task_waypoints.row(EndPointIndex); /* Implement here. Set the targetQ as
the final waypoint */
        }
        else // If the end point is not the last waypoint
        {
            //update the 'StartPointIndex' and the 'EndPointIndex' and reset the 'NodeIndex'
            StartPointIndex++; /* Implement here*/
            EndPointIndex++; /* Implement here*/
            NodeIndex = 0; /* Implement here*/
        }
    }
    else
    {
        NodeIndex++;
    }

    return targetQ;
}

// Return stack_waypoints
Eigen::MatrixXd DirectTeaching::get_stack_waypoints()
{
    return stack_waypoints;
}
```

그림 4. DirectTeaching.cpp 코드

이제 EE405_SystemIntegration/src directory의 modified_direct_teaching.cpp 파일을 구현하였다. Mode에 따라 적절한 trajectory generation을 할 수 있도록 구현하였다. 먼저 'task1_waypoints'와 'task2_waypoints'를 지정하였다. 'waypoints.csv' 파일의 첫번째 row를 'task1_waypoints'의 final point로 지정하고 'waypoints.csv' 파일의 두번째 row를 'task2_waypoints'의 final point로 지정하였다.

```
//for direct teaching
DirectTeaching direct_teaching(dof);
Eigen::Matrix<double,2,4> task1_waypoints; //waypoints for task1, row 0 : start position, row 1 : target position
Eigen::Matrix<double,2,4> task2_waypoints; //waypoints for task2, row 0 : start position, row 1 : target position
direct_teaching.initialization(initialQ);
task1_waypoints.row(0) = initialQ; /* Implement here. Initialize the start point of task1_waypoints*/
task2_waypoints.row(0) = initialQ; /* Implement here. Initialize the start point of task2_waypoints*/

Eigen::MatrixXd stack_waypoints = direct_teaching.get_stack_waypoints();
task1_waypoints.row(1) = stack_waypoints.row(1); /* Implement here. Set the final point of task1_waypoints as first row of 'waypoints.csv' */
task2_waypoints.row(1) = stack_waypoints.row(2); /* Implement here. Set the final point of task1_waypoints as second row of 'waypoints.csv'*/

int mode = 0; // 0 : tracking task1_waypoints, 1 : tracking task2_waypoints
```

그림 5. 'task1_waypoints'와 'task2_waypoints'를 지정한 모습

이렇게 지정한 이후 mode 값에 따라 해당하는 trajectory generation을 수행하여 동작하게 구현하였다. Mode값이 0일 경우 'task1_waypoints'를 따라가며 mode값이 1일 경우 'task2_waypoints'를 따라간다.

```
// ////////////////////////////////// Direct teaching mode //////////////////////////////////
if(mode == 0)
{
    // trajectory generation from task1_waypoints_.row(0)(current position) to task1_waypoints_.row(1)
    targetQ = direct_teaching.TrajectoryGeneration(task1_waypoints);
    int num_task_waypoints = task1_waypoints.rows();

    //define transition event (task1-> task2) : norm of error in position is less than certain threshold
    const double threshold = 0.01;
    Eigen::VectorXd error = currentQ - task1_waypoints.row(1).transpose(); /* Implement here. error is defined as 'currentQ - final point of task1_waypoints*/

    // mode transition event
    if(error.norm() < threshold /* Implement here. codition of mode transition */)
    {
        std::cout<<"mode change"<<std::endl;
        mode = 1;
        direct_teaching.clear();
        task2_waypoints.row(0) = currentQ; /* Implement here. update start point of task2_waypoints */
    }
}
else if(mode == 1)
{
    // trajectory generation from task1_waypoints_.row(0)(current position) to task1_waypoints_.row(1)
    targetQ = direct_teaching.TrajectoryGeneration(task2_waypoints);
}
// ////////////////////////////////// Direct teaching mode END //////////////////////////////////
```

그림 6. Direct teaching을 mode값에 따라 다르게 구현한 모습

최종적인 modified_direct_teaching.cpp 코드는 아래와 같다.

```
/**
 * @file modified_direct_teaching.cpp
 * @author Seo Wook Han (tjdnr7117@kaist.ac.kr)
 * @brief
 * @version 0.1
 * @date 2023-05-01
 *
 * @copyright Copyright (c) 2023
 *
 */

#include <iostream>
#include <chrono>
#include <Eigen/Dense>

#include "Controller/ArticulatedSystem.hpp"
#include "Common/DirectTeaching.hpp"

using namespace Eigen;
```

```
using namespace std;

#define SERVERPORT "4950"  // the port users will be connecting to

///////////////// IMPORTANT BELOW COMMAND //////////////////
// cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
// echo 1 | sudo tee /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
// https://emanual.robotis.com/docs/en/dxl/x/xc330-t288/#
///////////////// IMPORTANT BELOW COMMAND //////////////////

int main(int argc, char *argv[])
{
    // The manipulator has 4 DOF
    int dof = 4;

    // The control frequency is 100Hz
    double control_freq = 100;
    double dt = 1/control_freq;

    // Dynamixel setup, ID, position resolution, motor torque constant, etc.
    std::vector<uint8_t> dynamixel_id_set;
    std::vector<int32_t> position_resolution;
    std::vector<double>  motor_torque_constants;
    for (size_t i = 0; i < dof; i++)
    {
        dynamixel_id_set.emplace_back(i);
        position_resolution.emplace_back(POSITION_RESOLUTION);
        motor_torque_constants.emplace_back(MOTOR_TORQUE_CONSTANT);
    }

    // Initialize the Articulated_system,
    // We need a dof of the system, Operation Mode(Position, Velocity, Current),
    // USB Port information (for serical communication),
    // and dynamixel setup
    ArticulatedSystem       articulated_system(dof,       ArticulatedSystem::Mode::VELOCITY,
"/dev/ttyUSB0",
                                    dynamixel_id_set,                     position_resolution,
motor_torque_constants); //4 DOF manipulator

    /**
     * The member function articulated_system.GetJointAngle() returns the joint angles of the
robot system, which in this context refer to the motor angle.
     */
    Eigen::VectorXd currentQ = articulated_system.GetJointAngle();
    Eigen::VectorXd initialQ = articulated_system.GetJointAngle();

    // Declare targetQ(motor angle), targetQdot(motor velocity)
    Eigen::VectorXd targetQ(dof); targetQ.setZero();
    Eigen::VectorXd targetQdot(dof); targetQdot.setZero();

    /**
     * @brief time instant is initialized.
     * loop_start : represents the moment when the 'while' loop starts.
     * loop_end : represents the moment when the 'while' loop ends.
     * current_time : current time instant.
     * initial_time : The time instant when the program(=main function) starts.
     */
    std::chrono::system_clock::time_point loop_start= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point loop_end= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point current_time= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point initial_time= std::chrono::system_clock::now();

    // P gain.
    Eigen::MatrixXd Kp(dof, dof); Kp.setZero();

    //for direct teaching
    DirectTeaching direct_teaching(dof);
    Eigen::Matrix<double,2,4> task1_waypoints; //waypoints for task1, row 0 : start position,
row 1 : target position
    Eigen::Matrix<double,2,4> task2_waypoints; //waypoints for task2, row 0 : start position,
row 1 : target position
    direct_teaching.initialization(initialQ);
    task1_waypoints.row(0)  = initialQ;  /* Implement here. Initialize the start point of
task1_waypoints*/
    task2_waypoints.row(0)  = initialQ;  /* Implement here. Initialize the start point of
task2_waypoints*/
```

```
   Eigen::MatrixXd stack_waypoints = direct_teaching.get_stack_waypoints();
   task1_waypoints.row(1) = stack_waypoints.row(1); /* Implement here. Set the final point
of task1_waypoints as first row of 'waypoints.csv' */
   task2_waypoints.row(1) = stack_waypoints.row(2); /* Implement here. Set the final point
of task1_waypoints as second row of 'waypoints.csv'*/

   int mode = 0; // 0 : tracking task1_waypoints, 1 : tracking task2_waypoints

   while(1)
   {
      current_time= std::chrono::system_clock::now();
      auto                          loop_elasped_time_microsec                          =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - loop_start);
      auto                         total_elasped_time_microsec                          =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - initial_time);

      // For 100 Hz control loop.
      if(loop_elasped_time_microsec.count()>=dt*1e6)
      {
         double                            total_elasped_time_sec                        =
static_cast<double>(total_elasped_time_microsec.count())/1e6;
         loop_start = std::chrono::system_clock::now();

         currentQ = articulated_system.GetJointAngle();

         //      ////////////////////////////////////      Direct      teaching      mode
////////////////////////////////////
         if(mode == 0)
         {
            // trajectory generation from task1_waypoints_.row(0)(current position) to
task1_waypoints_.row(1)
            targetQ = direct_teaching.TrajectoryGeneration(task1_waypoints);
            int num_task_waypoints = task1_waypoints.rows();

            //define transition event (task1-> task2) : norm of error in position is less
than certain threshold
            const double threshold = 0.01;
            Eigen::VectorXd error = currentQ - task1_waypoints.row(1).transpose(); /*
Implement here. error is defined as 'currentQ - final point of task1_waypoints*/

            // mode transition event
            if(error.norm() < threshold /* Implement here. codition of mode transition */)
            {
               std::cout<<"mode change"<<std::endl;
               mode = 1;
               direct_teaching.clear();
               task2_waypoints.row(0) = currentQ; /* Implement here. update start point of
task2_waypoints */
            }
         }
         else if(mode == 1)
         {
            // trajectory generation from task1_waypoints_.row(0)(current position) to
task1_waypoints_.row(1)
            targetQ = direct_teaching.TrajectoryGeneration(task2_waypoints);
         }
         //    ////////////////////////////////////    Direct    teaching    mode    END
////////////////////////////////////

         ////////////////////////////////////  P  Position  control  USING  Velocity  Mode
////////////////////////////////////
         for (size_t i = 0; i < dof; i++)
         {
            Kp(i,i) = 4;
         }
         targetQdot = -Kp * (currentQ - targetQ);
         articulated_system.SetGoalVelocity(targetQdot); // velocity control mode
         //////////////////////////////////// P Position control USING Velocity Mode END
////////////////////////////////////

         loop_end= std::chrono::system_clock::now();
         auto                           one_step_calculation_time                        =
std::chrono::duration_cast<std::chrono::microseconds>(loop_end - loop_start);
      }
```

```
    }

    return 0;
}
```

그림 7. modified_direct_teaching.cpp 코드

코드를 모두 구현한 후 VSCode를 활용해 compiler를 GCC arm-linux-gnueabihf로, compile mode를 Release mode로 하여 build 하였다. Build한 executable을 실행하였고 예측한 동작을 제대로 수행하는 것을 확인할 수 있었다. 특히 최종 position을 잘 유지하는 것을 확인하였다.

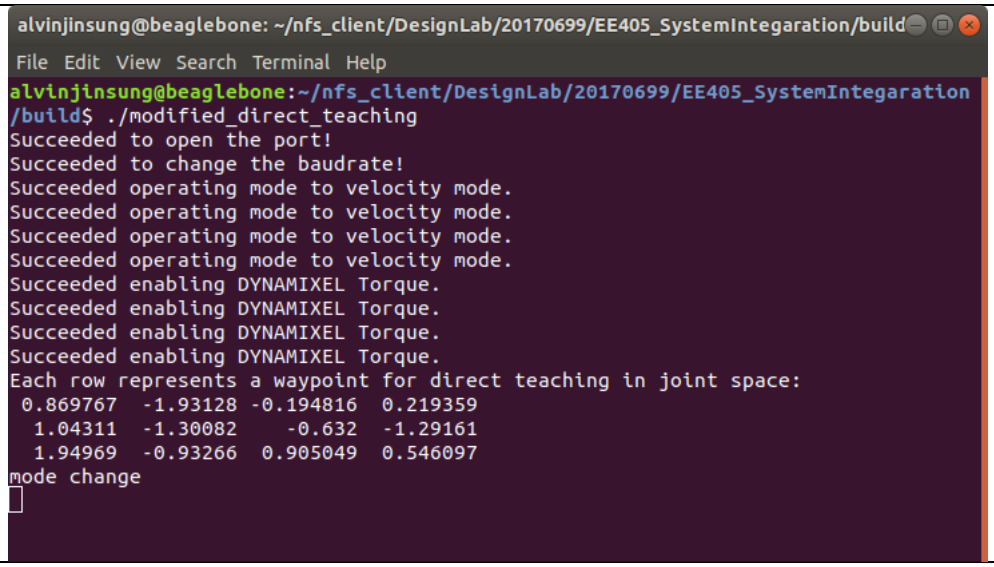


그림 8. Modified_direct_teaching을 실행한 terminal 창 모습

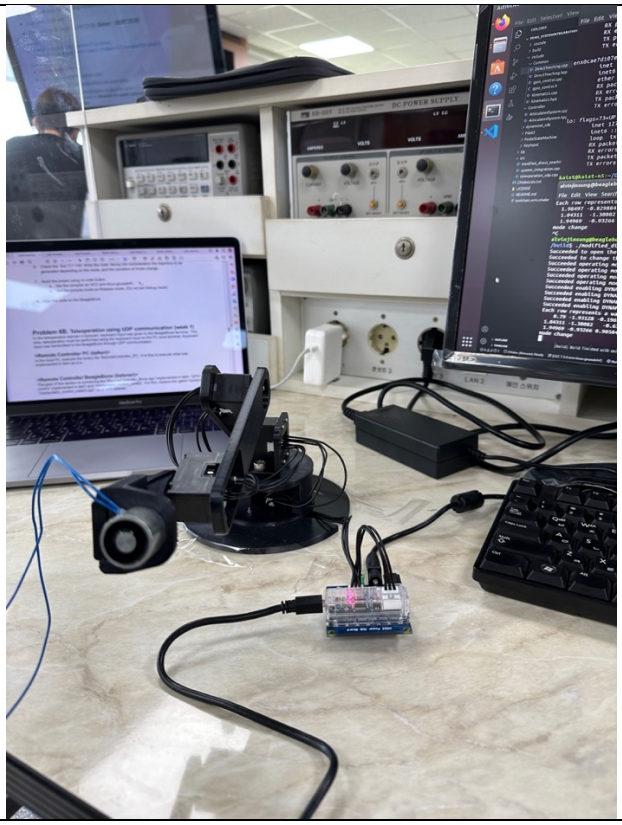


그림 9. Robot manipulator가 최종 position을 유지하는 모습

Problem 6B. Teleoperation using UDP communication

Lab 5에서 teleoperation을 위해 keyboard input을 Beaglebone terminal에서 준 것과 달리 이번 lab에서는 PC local terminal에서의 keyboard input을 통해 teleoperation을 수행할 것이다. 이를 위해 keyboard input이 UDP communication을 통해 Beaglebone으로 전송되어야 한다. PC에서는 lab 4에서 구현한 'RemoteController_PC'를 실행하여 talker 역할을 수행할 것이며 Beaglebone에서는 lab 4에서 구현한 'RemoteController_Bone.cpp', lab 2에서 구현한 'GPIO_control', 그리고 이번 lab에서 구현할 'manipulator_control_week2'를 통합하여 listener 역할을 수행할 것이다. 가장 먼저 EE405_SystemIntegration directory의 CMakeLists.txt파일을 problem 6B를 위한 executable build를 위해 수정하였다.

```
##for Problem2
add_library(EE405_SI_P2 STATIC include/Controller/ArticulatedSystem.cpp
                               include/Common/Kinematics.cpp
                               include/Common/gpio_control.cpp
                               )

add_executable(teleoperation_udp src/teleoperation_udp.cpp)
target_link_libraries(teleoperation_udp Threads::Threads EE405_SI_P2 dxl_sbc_cpp)
```

그림 10. CMakeLists.txt파일을 problem 6B를 위해 수정한 모습

다음으로 EE405_SystemIntegration/src directory에 있는 'teleoperation_udp.cpp' 파일을 구현하였다. Lab 4에서 구현했던 'RemoteController_Bone.cpp'를 참고하여 UDP communication에 필요한 variable들을 정의하고 UDP communication을 통해 receive한 data를 저장하였다. GetKeyBoardInput() function을 구현한 모습은 아래와 같다.

```
/*Implement here. Initialize all the variables required for udp communication*/
char buf;

int sockfd;
struct addrinfo hints, *servinfo, *p;
int rv;
int numbytes;
struct sockaddr_storage their_addr;
socklen_t addr_len;
char s[INET6_ADDRSTRLEN];

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_INET6; // set to AF_INET to use IPv4
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE; // use my IP

if ((rv = getaddrinfo(NULL, SERVERPORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return;
}

// loop through all the results and bind to the first we can
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("listener: socket");
        continue;
    }

    if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("listener: bind");
        continue;
    }

    break;
}

if (p == NULL) {
    fprintf(stderr, "listener: failed to bind socket\n");
    return;
}
```

그림 11. GetKeyBoardInput() function에서 UDP communication을 위한 variable을 정의하는 모습

```
while(1)
{
    printf("+-----------------------------------------------------+\n");
    printf("|   q: GPIO |  w: +y   |  e: stop |  r: +z  | i: init |\n");
    printf("|-----------+----------+----------+-----------------|\n");
    printf("|   a: -x   |  s: -y   |  d: +x   |  f: -z  |         |\n");
    printf("+-----------------------------------------------------+\n");

    /*Implement here. "Fill the 'buf' with the data received through UDP*/

    addr_len = sizeof their_addr;
    if ((numbytes = recvfrom(sockfd, &buf, sizeof(buf), 0, (struct sockaddr *)&their_addr, &addr_len)) == -1) {
        perror("recvfrom");
        exit(1);
    }

    std::cout<<"listener: packet contains "<<buf<<std::endl;

    key = buf;
    is_key_updated=true; // When a new keyboard input is received, is_key_upated is set to true
}

/*Implement here. close the socket*/

freeaddrinfo(servinfo);
close(sockfd);
```

그림 12. GetKeyBoardInput() function에서 Keyboard input을 통해 받은 data를 저장하는 모습

Keyboard input의 값에 따라 해당하는 action을 수행하야 한다. 지정된 keyboard input에 대한 action은 아래와 같다.

- r: move 1cm in the world z-axis
- f: move -1cm in the world z-axis
- w: move 1cm in the world y-axis
- s: move -1cm in the world y-axis
- d: move 1cm in the world x-axis
- a: move -1cm in the world x-axis
- i: move to the initial position
- e: end the teleoperation
- q: If the magnet is currently off, turn it on. If it is currently on, turn it off.

이를 main function내에서 구현한 모습은 아래와 같다.

```
////////////////////////////////// Teleoperation mode Start //////////////////////////////////
if(is_key_updated){ // Perform teleoperation only when a new keyboard input is received (is_key_upated=true)
    if(key=='i') // Move to the initial position for teleoperation
    {
        targetQ=initial_position;
        prev_targetQ=targetQ; // Update prev_targetQ
    }
    else if(key=='r' || key=='f' || key=='w' || key=='s' || key=='d' || key=='a') // Move the end-effector based on the task command
    {
        double position_resolution=0.01; // The position resolution for teleoperation is 1cm
        if(key == 'r') {
            task_command<< 0, 0, position_resolution; /*Implement here*/
        }
        else if(key == 'f') {
            task_command<< 0, 0, -position_resolution; /*Implement here*/
        }
        else if(key == 'w') {
            task_command<< 0, position_resolution, 0; /*Implement here*/
        }
        else if(key == 's') {
            task_command<< 0, -position_resolution, 0; /*Implement here*/
        }
        else if(key == 'd') {
            task_command<< position_resolution, 0, 0; /*Implement here*/
        }
        else if(key == 'a') {
            task_command<< -position_resolution, 0, 0; /*Implement here*/
        }
        Eigen::MatrixXd J=Kinematics::GetJacobianMatrix(currentQ); // J is the Jacobian matrix for linear velocity represented in the world frame
        targetQ = currentQ + (J.transpose()*(J*J.transpose()).inverse()) * task_command; /*Implement here*/
        prev_targetQ=targetQ; // Update prev_targetQ
    }
```

그림 13. Keyboard input에 따른 action을 설정한 모습

```
    else if(key=='q')
    {
        //When the key input is 'q', if the magnet is currently off, turn it on. If it is currently on, turn it off
        is_magnetic_on_ = !is_magnetic_on_; /*Implement here. */

        if(is_magnetic_on_ == true)
        {
            std::cout<<"Magnetic ON"<<std::endl;
        }
        else
        {
            std::cout<<"Magnetic OFF"<<std::endl;
        }

        gpio_fd_set_value(fd_gpio_30, is_magnetic_on_);

        targetQ=prev_targetQ;
    }
    else if(key=='e'){ // End teleoperation
        break;
    }
    else // When the wrong keyboard input is received, the targetQ is the privious targetQ value
    {
        targetQ=prev_targetQ;
    }

    is_key_updated=false; // Set is_key_updated to false in order to check if the new keyboard input is received
}
else{ // Before receiving the new keyboard input, the targetQ is the privious targetQ value
    targetQ=prev_targetQ;
}
////////////////////////////////// Teleoperation mode END //////////////////////////////////
```

그림 14. Keyboard input에 따른 action을 설정한 모습

최종적인 teleoperation_udp.cpp 코드는 아래와 같다.

```
/**
 * @file teleoperation_udp.cpp
 * @author Seo Wook Han (tjdnr7117@kaist.ac.kr)
 * @brief
 * @version 0.1
 * @date 2023-05-01
 *
 * @copyright Copyright (c) 2023
 *
 */

#include <iostream>
#include <fstream>
#include <memory>
#include <string>
#include <unistd.h>
#include <chrono>
#include <Eigen/Dense>
#include <csignal>
#include <thread>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/time.h>

#include "Controller/ArticulatedSystem.hpp"
#include "Common/Kinematics.hpp"
#include "Common/gpio_control.h"

using namespace Eigen;
using namespace std;

#define SERVERPORT "4950"   // the port users will be connecting to
```

```
///////////////// IMPORTANT BELOW COMMAND //////////////////
// cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
// echo 1 | sudo tee /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
// https://emanual.robotis.com/docs/en/dxl/x/xc330-t288/#
///////////////// IMPORTANT BELOW COMMAND //////////////////

// GetKeyboardInput() function : Compute the task command from the keyboard input for teleopeartion.
// Keyboard input should be received from the local PC shell.
//   You can copy and paste the code that is implemented in lab4 (RemoteController_PC_Skeleton.cpp)

bool is_key_updated = false;
Eigen::Vector3d task_command;
char key='i';
void GetKeyboardInput()
{
   // initialize variables and error print
   // Get argument of destination IP (argv) of Bone
   // Init datagram socket. You will use UDP network.

   /*Implement here. Initialize all the variables required for udp communication*/
   char buf;

   int sockfd;
        struct addrinfo hints, *servinfo, *p;
        int rv;
        int numbytes;
        struct sockaddr_storage their_addr;
        socklen_t addr_len;
        char s[INET6_ADDRSTRLEN];

        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_INET6; // set to AF_INET to use IPv4
        hints.ai_socktype = SOCK_DGRAM;
        hints.ai_flags = AI_PASSIVE; // use my IP

        if ((rv = getaddrinfo(NULL, SERVERPORT, &hints, &servinfo)) != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
                return;
        }

        // loop through all the results and bind to the first we can
        for(p = servinfo; p != NULL; p = p->ai_next) {
                if ((sockfd = socket(p->ai_family, p->ai_socktype,
                                p->ai_protocol)) == -1) {
                        perror("listener: socket");
                        continue;
                }

                if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
                        close(sockfd);
                        perror("listener: bind");
                        continue;
                }

                break;
        }

        if (p == NULL) {
                fprintf(stderr, "listener: failed to bind socket\n");
                return;
        }

   while(1)
   {
      printf("+-----------------------------------------------------+\n");
      printf("|   q: GPIO |  w: +y   |  e: stop |  r: +z  | i: init |\n");
      printf("|-----------+----------+----------+-------------------|\n");
      printf("|   a: -x   |  s: -y   |  d: +x   |  f: -z  |         |\n");
      printf("+-----------------------------------------------------+\n");

      /*Implement here. "Fill the 'buf' with the data received through UDP*/

      addr_len = sizeof their_addr;
      if ((numbytes = recvfrom(sockfd, &buf, sizeof(buf), 0, (struct sockaddr *)&their_addr,
```

```
&addr_len)) == -1) {
          perror("recvfrom");
          exit(1);
       }

       std::cout<<"listener: packet contains "<<buf<<std::endl;

       key = buf;
       is_key_updated=true; // When a new keyboard input is received, is_key_upated is set to
true
    }

    /*Implement here. close the socket*/

    freeaddrinfo(servinfo);
         close(sockfd);
}
int main(int argc, char *argv[])
{
    std::thread t1 = std::thread(GetKeyboardInput); // Create thread t1 to get the keyboard
input from the user

    // The manipulator has 4 DOF
    int dof = 4;

    // The control frequency is 100Hz
    double control_freq = 100;
    double dt = 1/control_freq;

    // Dynamixel setup, ID, position resolution, motor torque constant, etc.
    std::vector<uint8_t> dynamixel_id_set;
    std::vector<int32_t> position_resolution;
    std::vector<double>  motor_torque_constants;
    for (size_t i = 0; i < dof; i++)
    {
       dynamixel_id_set.emplace_back(i);
       position_resolution.emplace_back(POSITION_RESOLUTION);
       motor_torque_constants.emplace_back(MOTOR_TORQUE_CONSTANT);
    }

    // Initialize the Articulated_system,
    // We need a dof of the system, Operation Mode(Position, Velocity, Current),
    // USB Port information (for serical communication),
    // and dynamixel setup
    ArticulatedSystem       articulated_system(dof,       ArticulatedSystem::Mode::VELOCITY,
"/dev/ttyUSB0",
                                    dynamixel_id_set,                     position_resolution,
motor_torque_constants); //4 DOF manipulator

    /**
     * The member function articulated_system.GetJointAngle() returns the joint angles of the
robot system, which in this context refer to the motor angle.
     */
    Eigen::VectorXd currentQ = articulated_system.GetJointAngle();
    Eigen::VectorXd initialQ = articulated_system.GetJointAngle();

    // Declare targetQ(motor angle), targetQdot(motor velocity)
    Eigen::VectorXd targetQ(dof); targetQ.setZero();
    Eigen::VectorXd targetQdot(dof); targetQdot.setZero();

    bool is_ready_for_teleoperation=false;
    Eigen::VectorXd prev_targetQ(dof); // Declare prev_targetQ(the previous targetQ value)

    // The initial joint position for teleoperation is [0, 0, PI/2, -PI/2]. This configuraiton
is far from singularities.
    Eigen::VectorXd initial_position(dof); initial_position << 0, 0, M_PI/2, -M_PI/2;
    prev_targetQ = initial_position;

    /**
     * @brief time instant is initialized.
     * loop_start : represents the moment when the 'while' loop starts.
     * loop_end : represents the moment when the 'while' loop ends.
     * current_time : current time instant.
     * initial_time : The time instant when the program(=main function) starts.
     */
    std::chrono::system_clock::time_point loop_start= std::chrono::system_clock::now();
```

```
    std::chrono::system_clock::time_point loop_end= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point current_time= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point initial_time= std::chrono::system_clock::now();

    // P gain.
    Eigen::MatrixXd Kp(dof, dof); Kp.setZero();

    // The following variables are for trajectory generation before starting teleoperation
    int NodeIndex=0;
    int NodeNum=500;

    //initialize gpio port
    int fd_gpio_30;
    gpio_export(30);
    gpio_set_dir(30, 1);
    fd_gpio_30 = gpio_fd_open(30);
    gpio_fd_set_value(fd_gpio_30, 1);
    bool is_magnetic_on_ = false;

    while(1)
    {
        current_time= std::chrono::system_clock::now();
        auto                              loop_elasped_time_microsec                              =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - loop_start);
        auto                             total_elasped_time_microsec                              =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - initial_time);

        // For 100 Hz control loop.
        if(loop_elasped_time_microsec.count()>=dt*1e6)
        {
            double                               total_elasped_time_sec                              =
static_cast<double>(total_elasped_time_microsec.count())/1e6;
            loop_start = std::chrono::system_clock::now();

            currentQ = articulated_system.GetJointAngle();
            currentQ.resize(4);

            if(!is_ready_for_teleoperation){  //  Move  the  end-effector  to  initial_position
before starting teleoperation
                targetQ   =   (initial_position   -   initialQ)/NodeNum*NodeIndex+initialQ;   //
trajectory generation using the first-order polynomial
                NodeIndex++;
                if(NodeIndex>NodeNum) {
                    is_ready_for_teleoperation=true; // Now, it is ready for teleopeartion
                    std::cout<<"Start teleoperation!"<<std::endl;
                }
            }

            else{
                ////////////////////////////////////      Teleoperation      mode      Start
////////////////////////////////////
                if(is_key_updated){ // Perform teleoperation only when a new keyboard input is
received (is_key_upated=true)
                    if(key=='i') // Move to the initial position for teleoperation
                    {
                        targetQ=initial_position;
                        prev_targetQ=targetQ; // Update prev_targetQ
                    }
                    else if(key=='r' || key=='f' || key=='w' || key=='s' || key=='d' || key=='a')
// Move the end-effector based on the task command
                    {
                        double  position_resolution=0.01;  //  The  position  resolution  for
teleoperation is 1cm
                        if(key == 'r') {
                            task_command<< 0, 0, position_resolution; /*Implement here*/
                        }
                        else if(key == 'f') {
                            task_command<< 0, 0, -position_resolution; /*Implement here*/
                        }
                        else if(key == 'w') {
                            task_command<< 0, position_resolution, 0; /*Implement here*/
                        }
                        else if(key == 's') {
                            task_command<< 0, -position_resolution, 0; /*Implement here*/
                        }
                        else if(key == 'd') {
```

```
                        task_command<< position_resolution, 0, 0; /*Implement here*/
                    }
                    else if(key == 'a') {
                        task_command<< -position_resolution, 0, 0; /*Implement here*/
                    }
                    Eigen::MatrixXd J=Kinematics::GetJacobianMatrix(currentQ); // J is the
Jacobian matrix for linear velocity represented in the world frame
                    targetQ  =  currentQ  +  (J.transpose()*(J*J.transpose()).inverse())  *
task_command; /*Implement here*/
                    prev_targetQ=targetQ; // Update prev_targetQ
                }
                else if(key=='q')
                {
                    //When the key input is 'q', if the magnet is currently off, turn it on.
If it is currently on, turn it off
                    is_magnetic_on_ = !is_magnetic_on_; /*Implement here. */

                    if(is_magnetic_on_ == true)
                    {
                        std::cout<<"Magnetic ON"<<std::endl;
                    }
                    else
                    {
                        std::cout<<"Magnetic OFF"<<std::endl;
                    }

                    gpio_fd_set_value(fd_gpio_30, is_magnetic_on_);

                    targetQ=prev_targetQ;
                }
                else if(key=='e'){ // End teleoperation
                    break;
                }
                else // When the wrong keyboard input is received, the targetQ is the privious
targetQ value
                {
                    targetQ=prev_targetQ;
                }

                is_key_updated=false; // Set is_key_updated to false in order to check if
the new keyboard input is received
            }
            else{ // Before receiving the new keyboard input, the targetQ is the privious
targetQ value
                targetQ=prev_targetQ;
            }
            ////////////////////////////////////       Teleoperation       mode       END
////////////////////////////////////
        }

        ////////////////////////////////////  P  Position  control  USING  Velocity  Mode
////////////////////////////////////
        for (size_t i = 0; i < dof; i++)
        {
            Kp(i,i) = 4;
        }
        targetQdot = -Kp * (currentQ - targetQ); /* Implement here. You have to implement
joint space P position Controller */
        articulated_system.SetGoalVelocity(targetQdot); // velocity control mode
        //////////////////////////////////// P Position control USING Velocity Mode END
////////////////////////////////////
        loop_end= std::chrono::system_clock::now();
        auto                          one_step_calculation_time                          =
std::chrono::duration_cast<std::chrono::microseconds>(loop_end - loop_start);
      }
   }

   t1.detach(); //  allows the main function to wait until t1 completes its execution

   gpio_unexport(30);

   return 0;
}
```

그림 15. teleoperation_udp.cpp 코드

코드를 모두 구현한 후 VSCode를 활용해 compiler를 GCC arm-linux-gnueabihf로, compile mode를 Release mode로 하여 build 하였다. Build한 executable을 실행하였고 예측한 동작을 제대로 수행하는 것을 확인할 수 있었다. 구성한 회로에서 저항에 열이 나는 문제가 있었는데 저항 값을 조금 더 큰 것으로 교체하여 어느 정도 해당 문제를 해결할 수 있었다.

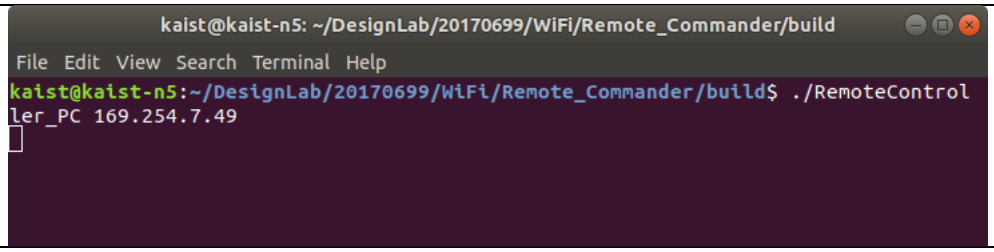


그림 16. PC terminal 창에서 RemoteController_PC을 실행하고 keyboard input을 넣은 모습

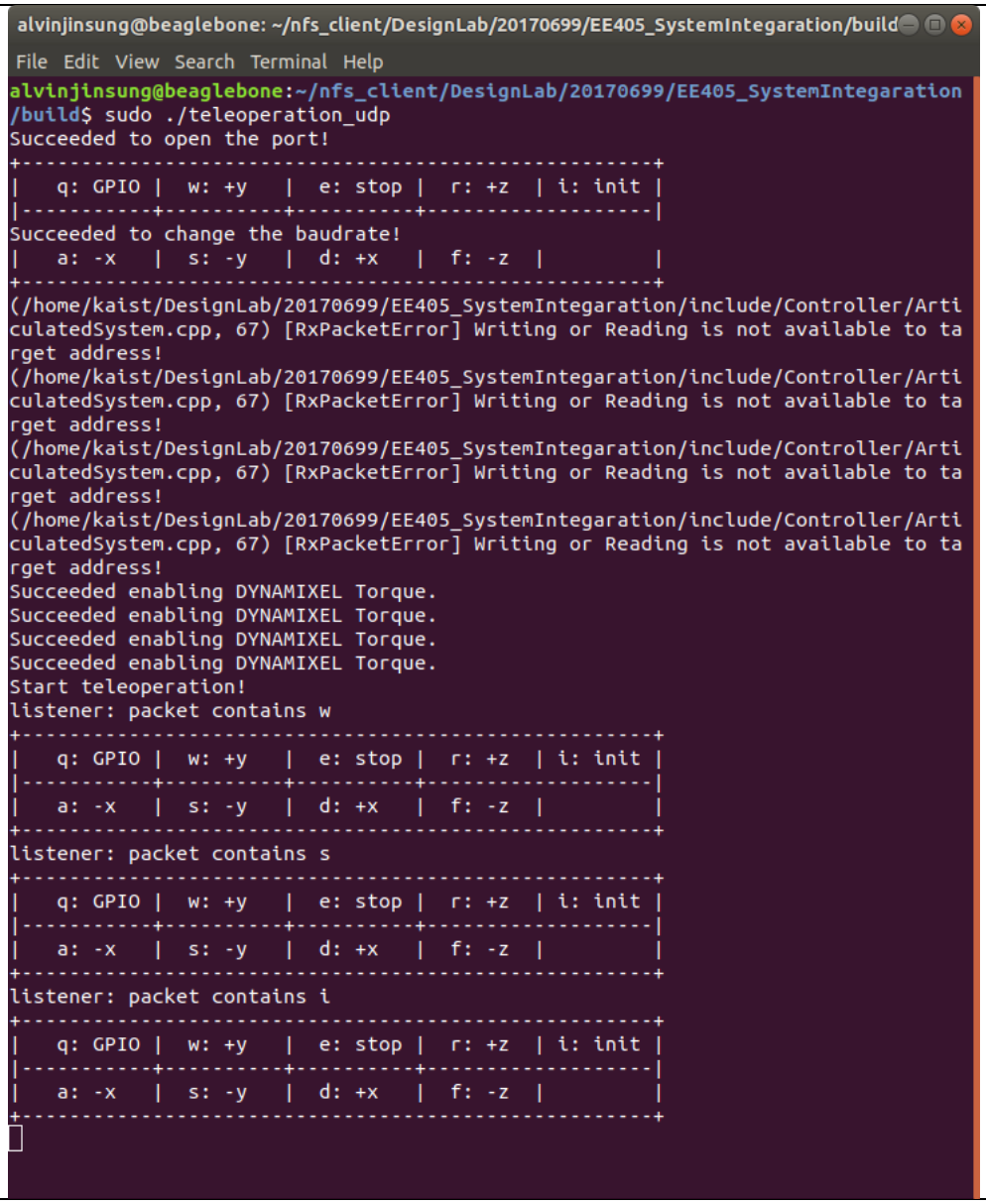


그림 17. Beaglebone terminal 창에서 teleoperation_udp를 실행한 모습

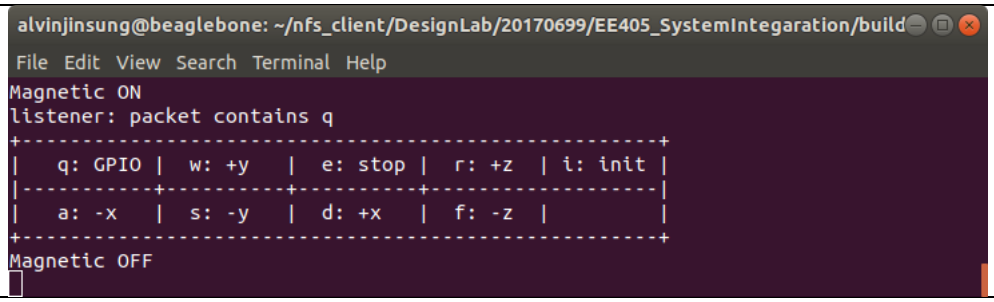


그림 18. Beaglebone terminal 창에서 teleoperation_udp를 실행한 모습

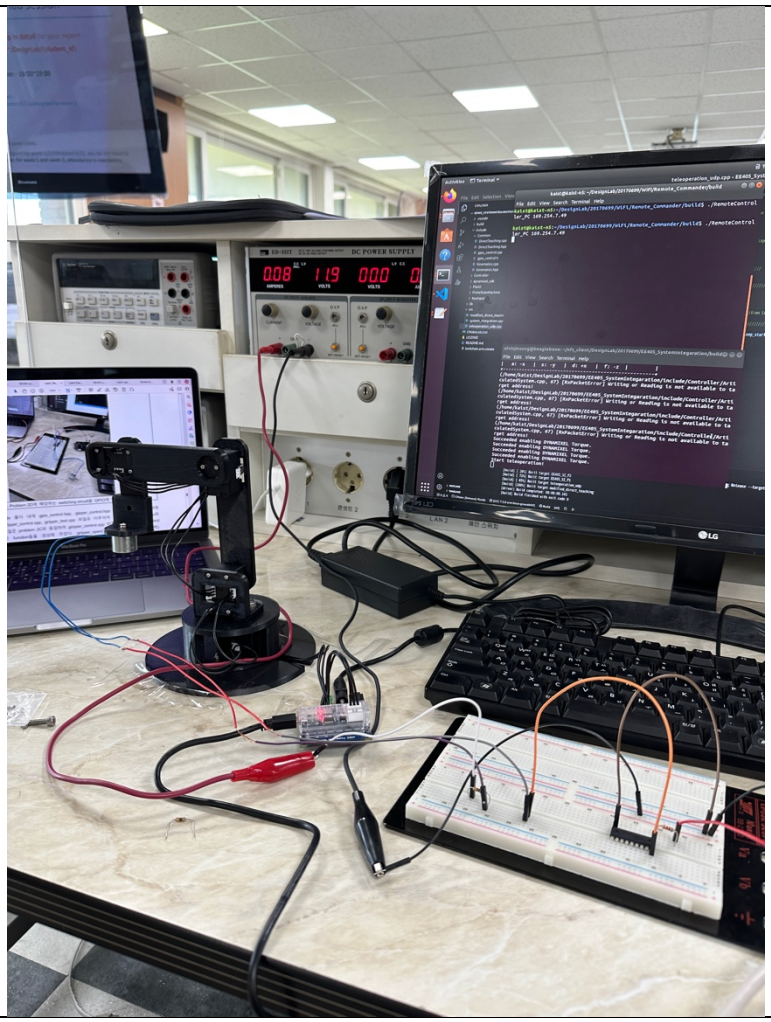


그림 19. Robot이 원하는 동작대로 움직이는 모습

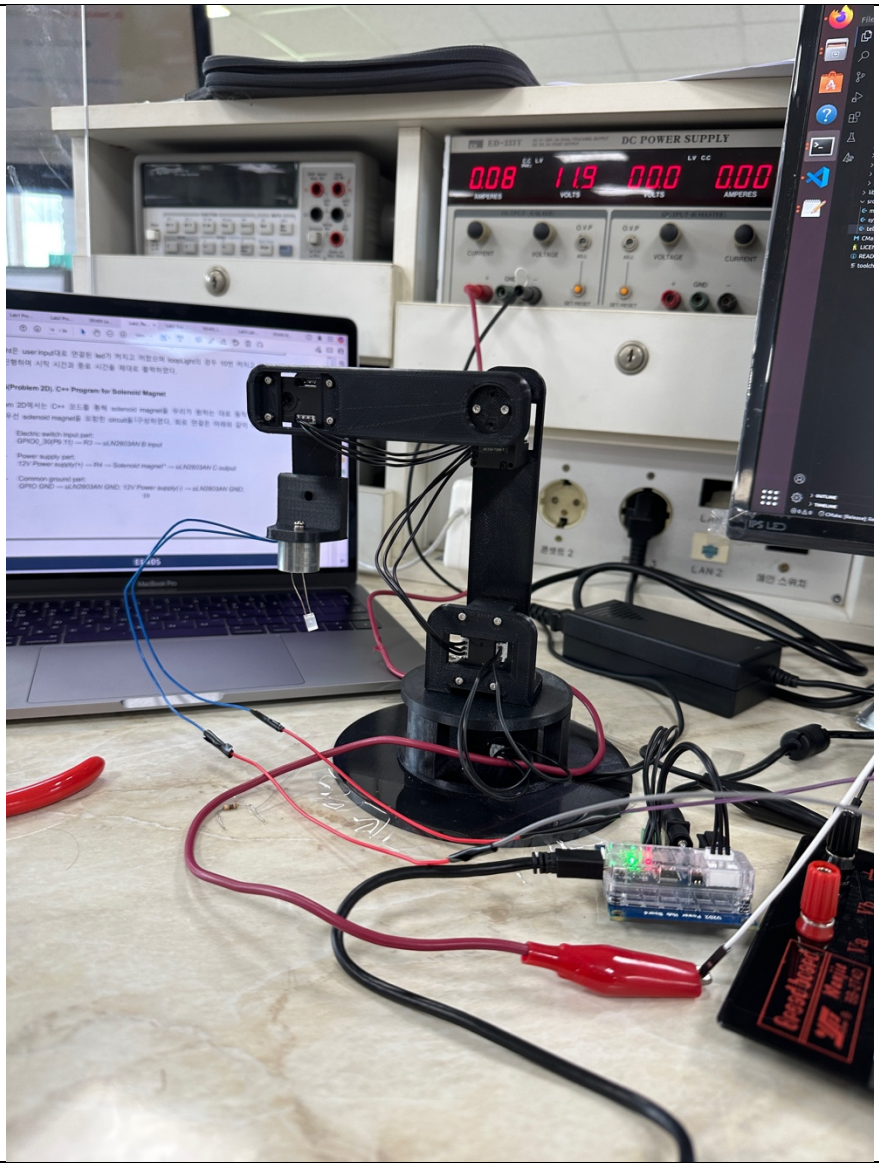


그림 20. Robot의 end-effector인 magnet이 켜진 모습

Problem 6C. Finite State Machine

Lab 6에서 구성하는 Finite State Machine System은 전체적으로 다음과 같이 설명할 수 있다. PC environment에서 Beaglebone에 연결되고 Robot을 보고 있는 Webcam을 통해 Robot의 상태를 streaming으로 확인할 수 있다. 이 때 적절한 keyboard input을 PC에서 입력하여 UDP communication을 통해 Beaglebone으로 전송, 해당 입력에 맞는 동작을 수행하게 된다. 기본적으로 미리 지정된 waypoint를 따라서 첫번째 waypoint로 이동 후 teleoperation을 통한 세부적인 제어 후 end-effector인 magnet을 켜 물체를 집고, 두번째 waypoint로 이동 후 다시 teleoperation을 통해 세부적인 제어 후 magnet을 꺼 물체를 놓게 된다. Keyboard input을 통해 어떤 task로 transition할지 조절할 수 있다. Webcam streaming을 통해 scene을 확인하며 keyboard input을 통해 제어하는 pick-and-place robot인 것이다. 제어에서는 direct teaching과 teleoperation을 이용해 상황에 따라 적절하게 사용한다.

가장 먼저 webcam streaming을 구성하기 위해 다음의 명령어들을 입력하였다

- Beaglebone
  $ ./capture -F -c 0 -o | ffmpeg -vcodec mjpeg -i pipe:0 -f mjpeg udp://224.0.0.1:1234

- PC
  $ ffplay -i udp://224.0.0.1:1234

다음 명령어를 입력하여 webcam streaming이 정상적으로 작동하는 것을 확인하였다.

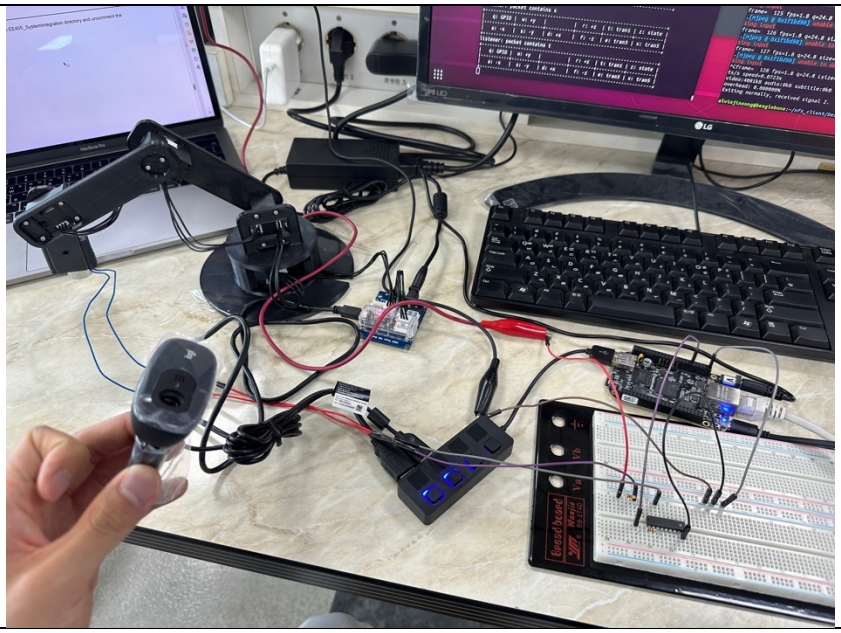


그림 21. Beaglebone에 webcam을 연결한 모습

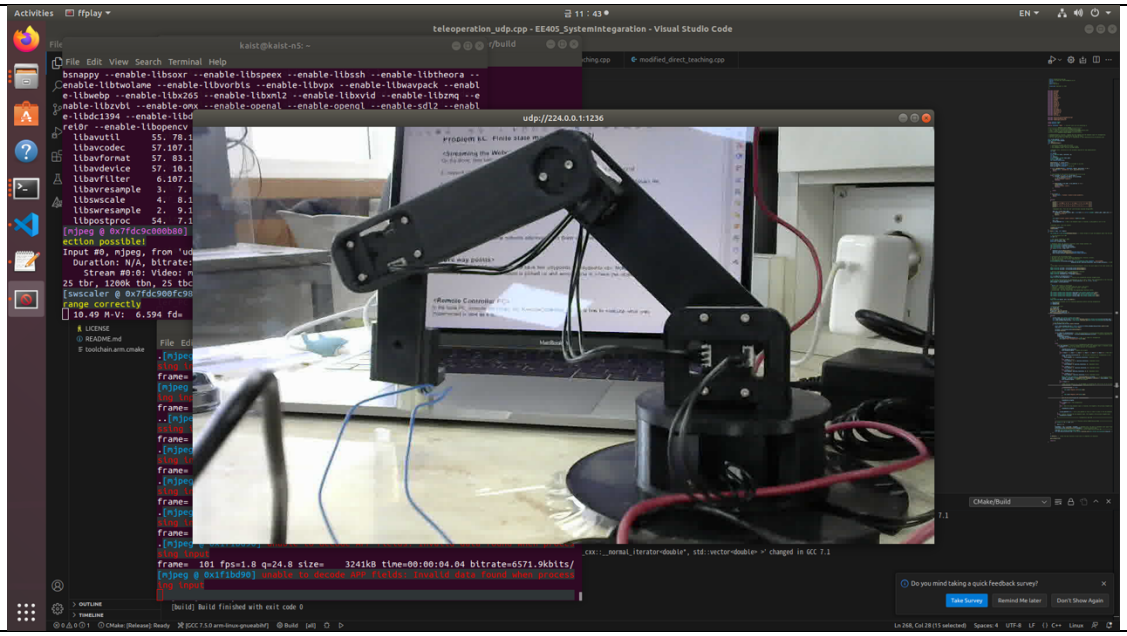


그림 22. Webcam Streaming이 정상적으로 작동하는 모습

다음으로 Finite state machine 구현을 위한 코드를 작성하였다. System_integration.cpp 파일에는 PC에서 입력된 keyboard input을 UDP communication을 통해 수신하여 변수에 저장하는 GetKeyboardInput() function과 finite state machine에서 어떤 state에 있는지에 따라 어떤 action을 취할지 지정되어 있는 main() function으로 구성되어 있다. 코드는 아래와 같다.

```
/**
 * @file system_integration.cpp
 * @author Seo Wook Han (tjdnr7117@kaist.ac.kr)
 * @brief
 * @version 0.1
 * @date 2023-05-01
 *
 * @copyright Copyright (c) 2023
 *
 */

#include <iostream>
#include <fstream>
#include <memory>
#include <string>
#include <unistd.h>
#include <chrono>
#include <Eigen/Dense>
#include <csignal>
#include <thread>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/time.h>

#include "Controller/ArticulatedSystem.hpp"
#include "KeyInput/getche.h"
#include "Common/Kinematics.hpp"
#include "FiniteStateMachine/fsm.h"
using namespace Eigen;
using namespace std;

#define SERVERPORT "4950"  // the port users will be connecting to

///////////////// IMPORTANT BELOW COMMAND //////////////////
// cat /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
// echo 1 | sudo tee /sys/bus/usb-serial/devices/ttyUSB0/latency_timer
// https://emanual.robotis.com/docs/en/dxl/x/xc330-t288/#
///////////////// IMPORTANT BELOW COMMAND //////////////////

// GetKeyboardInput() function : Compute the task command from the keyboard input for
teleopeartion.
// Keyboard input should be received from the local PC shell.
//   You   can   copy   and   paste   the   code   that   is   implemented   in   lab4
(RemoteController_PC_Skeleton.cpp)

bool is_key_updated = false;
Eigen::Vector3d task_command;
char key='i';
void GetKeyboardInput()
{
    // initialize variables and error print
    // Get argument of destination IP (argv) of Bone
    // Init datagram socket. You will use UDP network.

    /*Implement here. Initialize all the variables required for udp communication*/
    char buf;

    int sockfd;
         struct addrinfo hints, *servinfo, *p;
         int rv;
         int numbytes;
         struct sockaddr_storage their_addr;
         socklen_t addr_len;
         char s[INET6_ADDRSTRLEN];

         memset(&hints, 0, sizeof hints);
         hints.ai_family = AF_INET6; // set to AF_INET to use IPv4
```

```
        hints.ai_socktype = SOCK_DGRAM;
        hints.ai_flags = AI_PASSIVE; // use my IP

        if ((rv = getaddrinfo(NULL, SERVERPORT, &hints, &servinfo)) != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
                return;
        }

        // loop through all the results and bind to the first we can
        for(p = servinfo; p != NULL; p = p->ai_next) {
                if ((sockfd = socket(p->ai_family, p->ai_socktype,
                                p->ai_protocol)) == -1) {
                        perror("listener: socket");
                        continue;
                }

                if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
                        close(sockfd);
                        perror("listener: bind");
                        continue;
                }

                break;
        }

        if (p == NULL) {
                fprintf(stderr, "listener: failed to bind socket\n");
                return;
        }

   while(1)
   {
      printf("+--------------------------------------------------------------------+\n");
      printf("|   q: GPIO |  w: +y   |         |  r: +z  | t: tran1 | z: state |\n");
      printf("|-----------+----------+----------+---------+----------+----------|\n");
      printf("|   a: -x   |  s: -y   | d: +x    |  f: -z  | e: tran3 | x: tran5 |\n");
      printf("+--------------------------------------------------------------------+\n");

      /*Implement here. "Fill the 'buf' with the data received through UDP*/

      addr_len = sizeof their_addr;
      if ((numbytes = recvfrom(sockfd, &buf, sizeof(buf), 0, (struct sockaddr *)&their_addr,
&addr_len)) == -1) {
          perror("recvfrom");
          exit(1);
      }

      std::cout<<"listener: packet contains "<<buf<<std::endl;

      key = buf;
      is_key_updated=true; // When a new keyboard input is received, is_key_upated is set to
true
   }

   /*Implement here. close the socket*/

   freeaddrinfo(servinfo);
        close(sockfd);
}
int main(int argc, char *argv[])
{
   std::thread t1 = std::thread(GetKeyboardInput); // Create thread t1 to get the keyboard
input from the user

   // The manipulator has 4 DOF
   int dof = 4;

   // The control frequency is 100Hz
   double control_freq = 100;
   double dt = 1/control_freq;

   // Dynamixel setup, ID, position resolution, motor torque constant, etc.
   std::vector<uint8_t> dynamixel_id_set;
   std::vector<int32_t> position_resolution;
   std::vector<double>  motor_torque_constants;
   for (size_t i = 0; i < dof; i++)
```

```
    {
        dynamixel_id_set.emplace_back(i);
        position_resolution.emplace_back(POSITION_RESOLUTION);
        motor_torque_constants.emplace_back(MOTOR_TORQUE_CONSTANT);
    }

    // Initialize the Articulated_system,
    // We need a dof of the system, Operation Mode(Position, Velocity, Current),
    // USB Port information (for serical communication),
    // and dynamixel setup
    ArticulatedSystem       articulated_system(dof,       ArticulatedSystem::Mode::VELOCITY,
"/dev/ttyUSB0",
                                    dynamixel_id_set,                       position_resolution,
motor_torque_constants); //4 DOF manipulator

    /**
     * The member function articulated_system.GetJointAngle() returns the joint angles of the
robot system, which in this context refer to the motor angle.
     */
    Eigen::VectorXd currentQ = articulated_system.GetJointAngle();
    Eigen::VectorXd initialQ = articulated_system.GetJointAngle();

    // Declare targetQ(motor angle), targetQdot(motor velocity)
    Eigen::VectorXd targetQ(dof); targetQ.setZero();
    Eigen::VectorXd targetQdot(dof); targetQdot.setZero();

    /**
     * @brief time instant is initialized.
     * loop_start : represents the moment when the 'while' loop starts.
     * loop_end : represents the moment when the 'while' loop ends.
     * current_time : current time instant.
     * initial_time : The time instant when the program(=main function) starts.
     */
    std::chrono::system_clock::time_point loop_start= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point loop_end= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point current_time= std::chrono::system_clock::now();
    std::chrono::system_clock::time_point initial_time= std::chrono::system_clock::now();

    // P gain.
    Eigen::MatrixXd Kp(dof, dof); Kp.setZero();

    // Finite State Machine
    FSM fsm(dof);

    //initialize gpio port
    int fd_gpio_30;
    gpio_export(30);
    gpio_set_dir(30, 1);
    fd_gpio_30 = gpio_fd_open(30);
    gpio_fd_set_value(fd_gpio_30, 1);

    //initialize FSM
    fsm.initialization(initialQ);

    while(1)
    {
        current_time= std::chrono::system_clock::now();
        auto                            loop_elasped_time_microsec                          =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - loop_start);
        auto                           total_elasped_time_microsec                          =
std::chrono::duration_cast<std::chrono::microseconds>(current_time - initial_time);

        // For 100 Hz control loop.
        if(loop_elasped_time_microsec.count()>=dt*1e6)
        {
            double                              total_elasped_time_sec                      =
static_cast<double>(total_elasped_time_microsec.count())/1e6;
            loop_start = std::chrono::system_clock::now();
            currentQ = articulated_system.GetJointAngle();
            // std::cout<<"key : " << key << std::endl;
            switch(fsm.state)
            {
                case FSM_IDLE :
                    targetQ = fsm.Idle(key);
                    break;
                case FSM_TASK1 :
```

```
                targetQ = fsm.Task1(currentQ);
                break;
            case FSM_TASK2 :
                targetQ = fsm.Task2(key,is_key_updated, currentQ, fd_gpio_30);
                break;
            case FSM_TASK3 :
                targetQ = fsm.Task3(currentQ);
                break;
            default:
                targetQ = fsm.Idle(key);
                break;
        }

        if(is_key_updated==true)
        {
            if(key == 'z'){fsm.print_state();}
            is_key_updated = false;
        }

        ////////////////////////////////////  P  Position  control  USING  Velocity  Mode
//////////////////////////////////
        for (size_t i = 0; i < dof; i++)
        {
            Kp(i,i) = 4;
        }
        // targetQdot =/* Implement here. You have to implement P position Controller */
        targetQdot = -Kp * (currentQ - targetQ);
        articulated_system.SetGoalVelocity(targetQdot); // velocity control mode
        //////////////////////////////////// P Position control USING Velocity Mode END
//////////////////////////////////

        loop_end= std::chrono::system_clock::now();
        auto                          one_step_calculation_time                          =
std::chrono::duration_cast<std::chrono::microseconds>(loop_end - loop_start);
      }

    }

    t1.detach(); //  allows the main function to wait until t1 completes its execution

    return 0;
}
```

그림 23. system_integration.cpp 코드

위 코드에서 살펴보면 enum으로 설정된 finite state macine의 stated에 따라 각기 다른 fsm 코드의 함수들이 호출되는 것을 볼 수 있다. 해당 function들이 정의된 include/FiniteStateMachine directory에 fsm.cpp 파일을 완성하였다. 우리가 구현하려는 finite state machine은 아래와 같다.

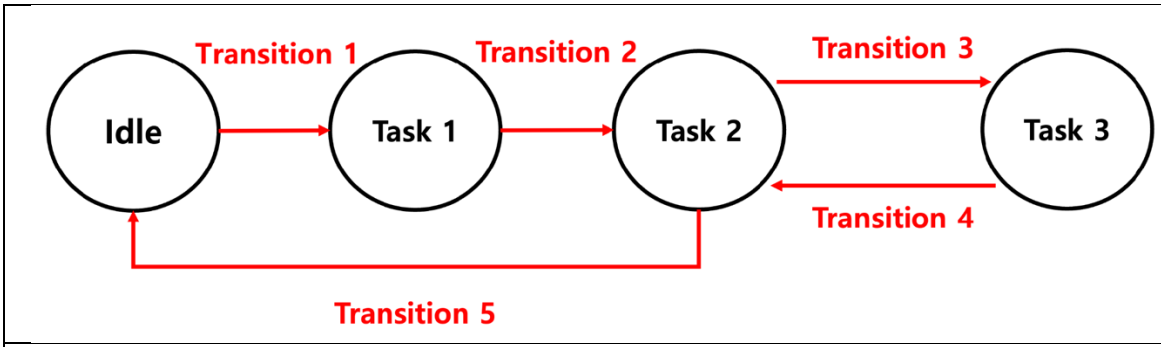


그림 24. Lab 6의 finite state machine

States:

- Idle: The robot maintains its current position.
- Task 1: Using a direct teaching method, move the end-effector of the robot to near the target object.
- Task 2: Using the keyboard input from Ubuntu PC, control the end-effector position of the robot and turn on/off the solenoid magnet to pick or place a target object. This process must use visual

feedback through a camera.
- Task 3: Using a direct teaching method, move the end-effector to the position where the object is to be placed.

Transition:
- Transition 1: When the keyboard input is ‘t’.
- Transition 2: The error norm between q and last way point is less than a certain threshold.
- Transition 3: When the keyboard input is ‘e’. (When object pick is success, the user gives the input ‘e’)
- Transition 4: The error norm between q and last way point is less than a certain threshold.
- Transition 5: When the keyboard input is ‘x’. (When object place is success, the user gives the input ‘x’)

각 state마다 다른 action을 하게 되며, fsm.cpp 코드 상에서 각 state별 실행 함수의 output은 desired joint position이다. 완성한 fsm.cpp 코드는 아래와 같다.

```
#include "fsm.h"

FSM::FSM(const int & dof) : dof_(dof), direct_teaching_(dof){}

void FSM::initialization(const Eigen::VectorXd &current_q)
{
    //initialize Task1,3 (direct teaching)
    /* Implement here.*/

    direct_teaching_.initialization(current_q);

    task1_waypoints_.row(0) = current_q;
    task2_waypoints_.row(0) = current_q;

    prev_targetQ = current_q;

    Eigen::MatrixXd stack_waypoints = direct_teaching_.get_stack_waypoints();

    task1_waypoints_.row(1) = stack_waypoints.row(1);
    task2_waypoints_.row(1) = stack_waypoints.row(2);
}

void FSM::print_state()
{
    std::cout<<"Current state is ";
    switch(state)
    {
        case FSM_IDLE :
            std::cout<<"FSM_IDLE"<<std::endl;
            break;
        case FSM_TASK1 :
            std::cout<<"FSM_TASK1"<<std::endl;
            break;
        case FSM_TASK2 :
            std::cout<<"FSM_TASK2"<<std::endl;
            break;
        case FSM_TASK3 :
            std::cout<<"FSM_TASK3"<<std::endl;
            break;
    }
}

Eigen::VectorXd FSM::Idle(const char &key_input)
{
    //define Idle
    Eigen::VectorXd targetQ(dof_); targetQ.setZero();
    /* Implement here.*/

    //define transition event
    /* Implement here.*/

    targetQ = prev_targetQ;

    if (key_input == 't') {
        state = FSM_TASK1;
    }
```

```
    prev_targetQ=targetQ;

    return targetQ;
}

// tracking task1_waypoints
Eigen::VectorXd FSM::Task1(const Eigen::VectorXd &current_q)
{
    //define Task1
    Eigen::VectorXd targetQ(dof_); targetQ.setZero();
    /* Implement here.*/

    targetQ = direct_teaching_.TrajectoryGeneration(task1_waypoints_);

    Eigen::VectorXd error = current_q-task1_waypoints_.row(1).transpose();

    if(error.norm() < 0.01){
        direct_teaching_.clear();
        task2_waypoints_.row(0) = current_q;

        state = FSM_TASK2;
    }

    prev_targetQ=targetQ;

    //define transition event
    /* Implement here.*/

    return targetQ;
}

// task space control mode
Eigen::VectorXd  FSM::Task2(const  char  &key_input,  const  bool&  is_key_updated,  const
Eigen::VectorXd &current_q, const int &fd_gpio_30)
{
    //define Task2
    Eigen::VectorXd targetQ(dof_); targetQ.setZero();
    /* Implement here.*/

    Eigen::Vector3d task_command;

    ////////////////////////////////////          Teleoperation          mode          Start
////////////////////////////////////
    if(is_key_updated){ // Perform teleoperation only when a new keyboard input is received
(is_key_upated=true)
        if(key_input=='i') // Move to the initial position for teleoperation
        {
            targetQ=current_q;
            prev_targetQ=targetQ; // Update prev_targetQ
        }
        else  if(key_input=='r'  ||  key_input=='f'  ||  key_input=='w'  ||  key_input=='s'  ||
key_input=='d' || key_input=='a') // Move the end-effector based on the task command
        {
            double position_resolution=0.01; // The position resolution for teleoperation is
1cm
            if(key_input == 'r') {
                task_command<< 0, 0, position_resolution; /*Implement here*/
            }
            else if(key_input == 'f') {
                task_command<< 0, 0, -position_resolution; /*Implement here*/
            }
            else if(key_input == 'w') {
                task_command<< 0, position_resolution, 0; /*Implement here*/
            }
            else if(key_input == 's') {
                task_command<< 0, -position_resolution, 0; /*Implement here*/
            }
            else if(key_input == 'd') {
                task_command<< position_resolution, 0, 0; /*Implement here*/
            }
            else if(key_input == 'a') {
                task_command<< -position_resolution, 0, 0; /*Implement here*/
            }
            Eigen::MatrixXd J=Kinematics::GetJacobianMatrix(current_q); // J is the Jacobian
matrix for linear velocity represented in the world frame
```

```
            targetQ = current_q + (J.transpose()*(J*J.transpose()).inverse()) * task_command;
/*Implement here*/
            prev_targetQ=targetQ; // Update prev_targetQ
        }
        else if(key_input=='q')
        {
            //When the key input is 'q', if the magnet is currently off, turn it on. If it is
currently on, turn it off
            is_magnetic_on_ = !is_magnetic_on_; /*Implement here. */

            if(is_magnetic_on_ == true)
            {
                std::cout<<"Magnetic ON"<<std::endl;
            }
            else
            {
                std::cout<<"Magnetic OFF"<<std::endl;
            }

            gpio_fd_set_value(fd_gpio_30, is_magnetic_on_);

            targetQ=prev_targetQ;
        }
        else if(key_input=='e'){ // End teleoperation
            task2_waypoints_.row(0) = current_q;
            state = FSM_TASK3;
        }
        else if(key_input=='x'){ // End teleoperation
            task1_waypoints_.row(0) = current_q;
            state = FSM_IDLE;
        }
        else // When the wrong keyboard input is received, the targetQ is the privious targetQ
value
        {
            targetQ=prev_targetQ;
        }
    }
    else{ // Before receiving the new keyboard input, the targetQ is the privious targetQ
value
        targetQ=prev_targetQ;
    }
    ////////////////////////////////////          Teleoperation          mode          END
////////////////////////////////////

    //define transition event
    /* Implement here.*/

    return targetQ;
}

// tracking task2_waypoints
Eigen::VectorXd FSM::Task3(const Eigen::VectorXd &current_q)
{
    //define Task3
    Eigen::VectorXd targetQ(dof_); targetQ.setZero();
    /* Implement here.*/

    targetQ = direct_teaching_.TrajectoryGeneration(task2_waypoints_);

    Eigen::VectorXd error = current_q - task2_waypoints_.row(1).transpose();

    if(error.norm() < 0.01){
        direct_teaching_.clear();

        state = FSM_TASK2;
    }

    prev_targetQ = targetQ;

    //define transition event
    /* Implement here.*/

    return targetQ;
}
```

그림 25. fsm.cpp 코드

위의 코드들을 build하여 실행 executable을 생성하기 위해 CMakeLists.txt 파일을 적절하게 수정하고 VSCode를 이용하여 build하였다.

```
#for Problem3
add_library(FSM STATIC include/FiniteStateMachine/fsm.cpp)
target_link_libraries(FSM EE405_SI_P1)

add_executable(system_integration src/system_integration.cpp)
target_link_libraries(system_integration Threads::Threads EE405_SI_P1 FSM dxl_sbc_cpp)
```

그림 26. CMakeLists.txt파일을 problem 6C를 위해 수정한 모습

PC에서는 RemoteController_PC를 실행시키고 Beaglebone에서는 build된 system_integration을 실행한 결과 PC에서의 keyboard input에 따라 우리가 구상한 finite state machine의 동작을 적절히 수행하는 모습을 보여주었다.

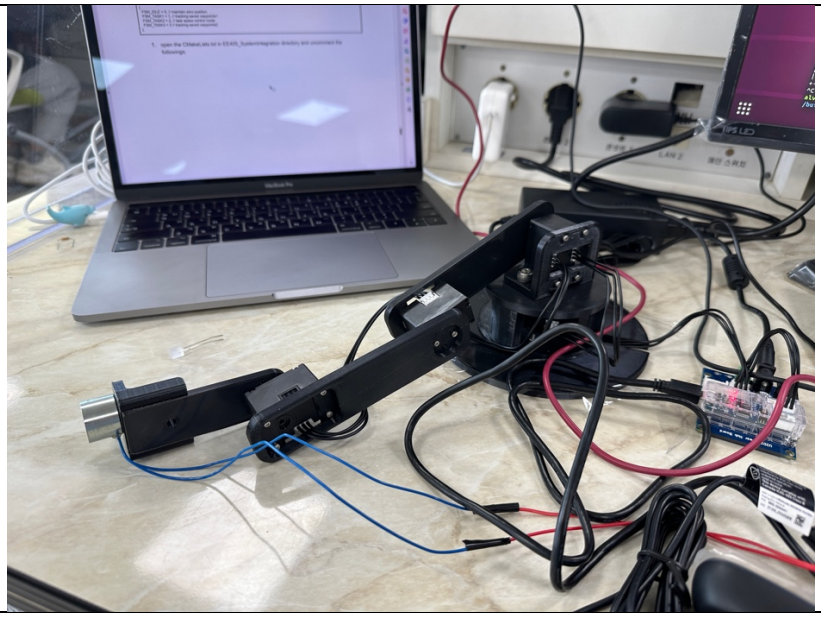


그림 27. Initial 상태의 robot manipulator(FSM_IDLE state)

그림 28. ‘t’ 입력시 direct teaching method으로 첫번째 waypoint로 이동, error norm이 threshold 보다 작을 경우 해당 position 유지(FSM_TASK1 state)

그림 29. Teleoperation을 통해 end-effector position 이동 후 magnet을 켜 물체를 pick (FSM_Task2 state)

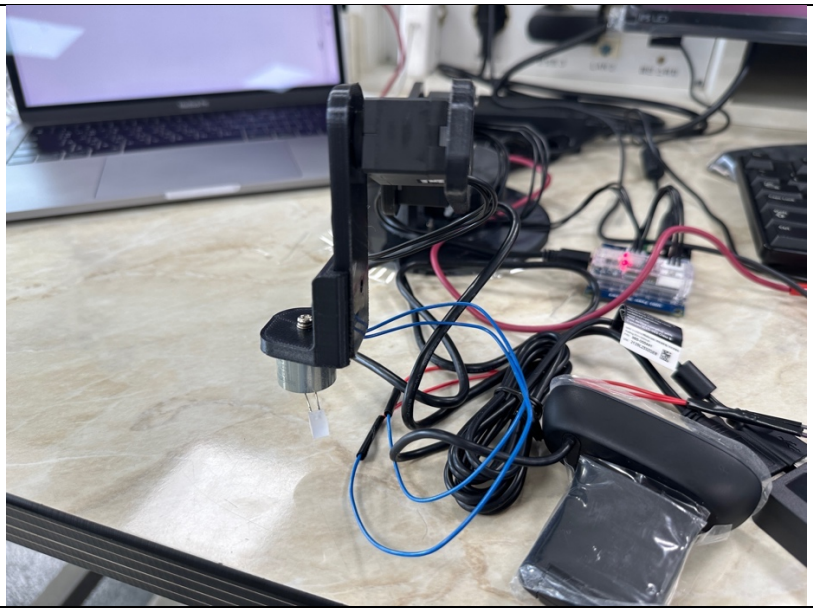


그림 30. ‘e’ 입력시 direct teaching method으로 두번째 waypoint로 이동, error norm이 threshold 보다 작을 경우 해당 position 유지(FSM_TASK3 state)

그림 31. Teleoperation을 통해 end-effector position 이동 후 magnet을 꺼 물체를 place (FSM_Task2 state)

실험을 통해 원하는 동작들을 해당 state에 맞추어 수행하였으며 transition 조건들도 모두 만족해 구상한 finite state machine을 성공적으로 구현하였다는 것을 확인하였다.

Problem 6D. Choose Your Own Problem

Problem Definition

우리가 구성한 pick-and-place robot의 경우 산업 현장에서 많이 사용되게 된다. 그리고 산업 현장에서의 안전은 항상 중요한 부분이다. Robot manipulator에서 안정상 신경 써야 할 부분은 robot에 너무 큰 부하나 외부적인 힘이 걸리지는 않는지, motor의 적절한 교체 시기 등이다. System specification을 확인하게 되면 motor나 end-effector가 견딜 수 있는 힘의 크기를 확인할 수 있으나 이것을 실제 현장에서 즉각적으로 확인하기 쉽지 않다. 그렇기에 너무 큰 부하가 걸렸을 시 알림(led 점등, 알람 등)의 방법으로 안전상 위험한 상태라는 것을 자동으로 확인하는 것이 중요한 역할을 할 수 있다. 또한 motor에 가해진 누적된 부하의 크기를 기록하고 tracking 함으로써 motor의 적절한 점검 및 교체 시기를 파악할 수 있다. 따라서 나는 pick-and-place robot의 motor에 걸리는 부하가 특정 threshold 이상일 때 자동으로 알림(led 점등)을 통해 안정상 위험상황을 표시하는 system을 구축하고 motor에 가해진 누적된 힘의 크기를 기록함으로써 motor 점검 및 교체 시기를 쉽게 파악 가능한 system을 설계하고자 한다.

Methodology

우선 motor에 가해지는 torque 크기를 ArticulatedSystem에서 GetTorqueStates() 함수를 이용해 얻었다.

```
Eigen::VectorXd ArticulatedSystem::GetTorqueStates()
{

    // Syncread present position
    dxl_comm_result = groupSyncReads.at(2).txRxPacket();
    if (dxl_comm_result != COMM_SUCCESS)
    {
        printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
    }
    else
    {
        for (size_t i = 0; i < dof; i++)
        {
            if (groupSyncReads.at(2).getError(dynamixel_id_set.at(i), &dxl_error))
            {
                printf("[ID:%03d] %s\n", dynamixel_id_set.at(i), packetHandler->getRxPacketError(dxl_error));
            }
        }
    }

    for (size_t i = 0; i < dof; i++)
    {
        // Check if groupsyncread data of Dynamixel#1 is available
        dxl_getdata_result = groupSyncReads.at(2).isAvailable(dynamixel_id_set.at(i), ADDR_PRESENT_CURRENT, LEN_GOAL_CURRENT);
        if (dxl_getdata_result != true)
        {
            fprintf(stderr, "[%s, %d][ID:%03d] groupSyncRead getdata failed\n", __FILE__, __LINE__, dynamixel_id_set.at(i));
        }
    }

    for (size_t i = 0; i < dof; i++)
    {
        int16_t qcurrent_counts = groupSyncReads.at(2).getData(dynamixel_id_set.at(i), ADDR_PRESENT_CURRENT, LEN_GOAL_CURRENT);
        if (i==0) qtorque(i) = (mA_PER_COUNT/1000.0)*motor_torque_constants[i]*qcurrent_counts;
        else qtorque(i) = (1/1000.0)*motor_torque_constants[i]*qcurrent_counts;
    }
    return qtorque;
}
```

그림 32. ArticulatedSystem.cpp 파일의 GetTorqueStates() function

여기서는 motor_torque_constants를 통해 current에서 torque를 계산해 내는데 이는 motor의 system specification에서 torque-current graph에서 기울기를 통해 얻어 입력하였다. Motor는 dynamixel XC330-T288-T이며 performance graph는 아래와 같다.

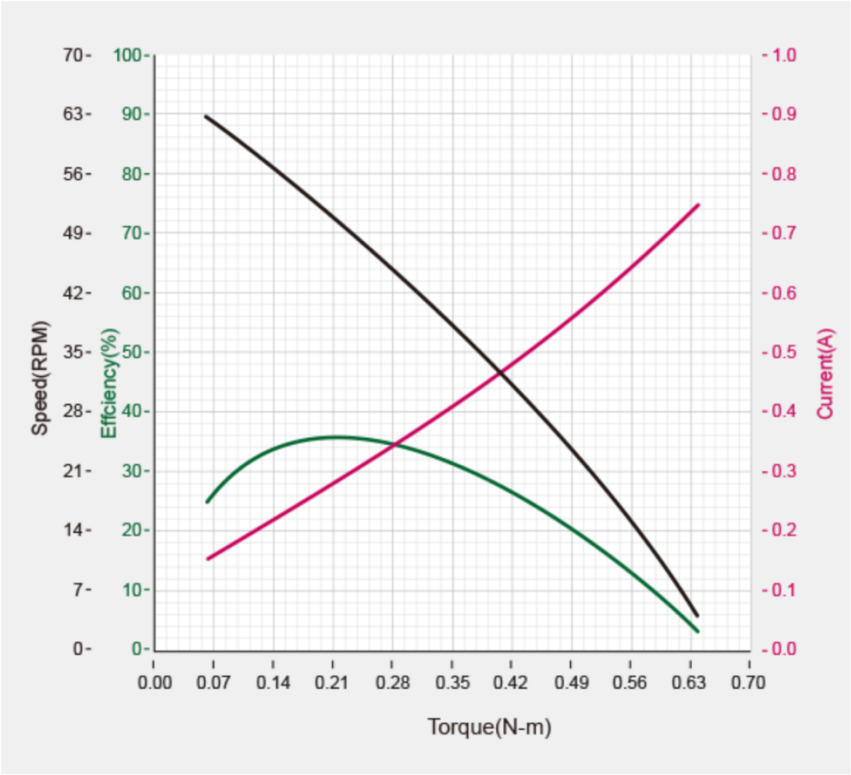


그림 33. dynamixel XC330-T288-T performance graph

이제 motor의 torque 값이 특정 threshold를 넘었을 때 알림으로 led 점등을 위해 led 회로를 구성하고 Beaglebone GPIO31에 연결하였다. 연결된 모습은 아래와 같다.

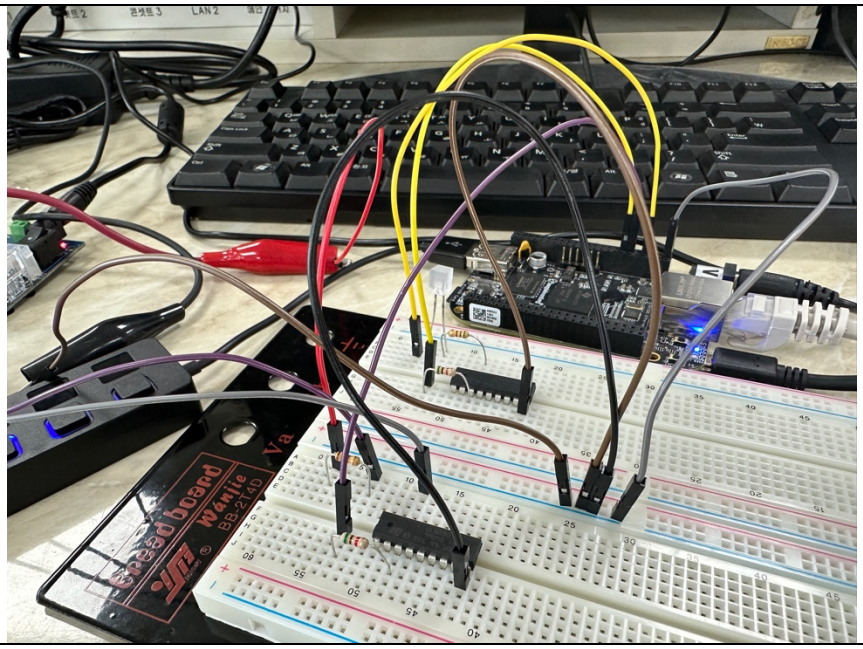


그림 34. LED를 Beaglebone에 연결한 모습(GPIO31)

해당 led 동작을 제어하기 위해 GPIO 31번 포트를 적절하게 initialization 해주고 동작 확인 및 manual하게 알림 시스템을 동작 시켜야 할 때를 대비해 keyboard input을 추가하여 'l' 입력시 led가 키고 꺼지게 코드를 구현하였다.

```
//initialize gpio port
int fd_gpio_30;
gpio_export(30);
gpio_set_dir(30, 1);
fd_gpio_30 = gpio_fd_open(30);
gpio_fd_set_value(fd_gpio_30, 1);

int fd_gpio_31;
gpio_export(31);
gpio_set_dir(31, 1);
fd_gpio_31 = gpio_fd_open(31);
gpio_fd_set_value(fd_gpio_31, 1);
```

그림 35. GPIO port 31번을 initialization 해주는 모습

```
//Finite state machine
class FSM
{
    private:
    //common variables
    const int dof_;

    //For Task1,3 (Direct teaching)
    DirectTeaching direct_teaching_;
    Eigen::Matrix<double,2,4> task1_waypoints_; //waypoints for task1, row 0 : start position, row 1 : target position
    Eigen::Matrix<double,2,4> task2_waypoints_; //waypoints for task2, row 0 : start position, row 1 : target position

    //For Task2 (task sapce control)
    Eigen::VectorXd prev_targetQ;
    bool is_magnetic_on_ = false;
    bool is_light_on_ = false;

    public:
    FSM_state state = FSM_IDLE; // set initial state as idel.

    FSM(const int & dof);

    void initialization(const Eigen::VectorXd &current_q);
    void print_state();
    // maintain current position
    Eigen::VectorXd Idle(const char &key_input);
    // tracking saved waypoints
    Eigen::VectorXd Task1(const Eigen::VectorXd &current_q);
    // task space control mode
    Eigen::VectorXd Task2(const char &key_input, const bool& is_key_updated, const Eigen::VectorXd &q, const int &fd_gpio_30, const int &fd_gpio_31);
    // tracking saved waypoints in reverse order
    Eigen::VectorXd Task3(const Eigen::VectorXd &current_q);


};
```

그림 36. fsm_own.hpp 파일에 teleoperation state 함수에 fd_gpio_31 변수를 추가한 모습

```
else if(key_input=='l')
{
    //When the key input is 'l', if the light is currently off, turn it on. If it is currently on, turn it off
    is_light_on_ = !is_light_on_; /*Implement here. */

    if(is_light_on_ == true)
    {
        std::cout<<"light ON"<<std::endl;
    }
    else
    {
        std::cout<<"light OFF"<<std::endl;
    }

    gpio_fd_set_value(fd_gpio_31, is_light_on_);

    targetQ=prev_targetQ;
}
```

그림 37. Fsm.cpp 파일에 keyboard input 'l'에 대한 동작을 추가한 모습

이를 build하여 실행시켜 보았고 정상적으로 led가 keyboard input 'l'에 의해 켜졌다 꺼지는 것을 확인할 수 있었다. 이제 system_integration_own.cpp 파일에서 ArticulatedSystem.GetTorqueValue()를 통해 얻은 각 motor 값이 특정 threshold 이상일 경우 led가 켜지도록 코드를 구현하였으며 누적 torque값을 계산해 기록하도록 하였다.

```
Eigen::VectorXd current_torque = articulated_system.GetTorqueStates();
torque_total = torque_total + current_torque;

std::cout<<"torque current: "<<current_torque<<std::endl;
std::cout<<"torque total: "<<torque_total<<std::endl;

if(current_torque(0) > 0.06 || current_torque(1) > 0.06 || current_torque(2) > 0.06 || current_torque(3) > 0.06) {
    gpio_fd_set_value(fd_gpio_31, 1);
}

else {
    gpio_fd_set_value(fd_gpio_31, 0);
}
```

그림 38. System_integration_own.cpp 파일에 torque 값에 따라 led 점등을 제어하고 torque 누적 합계를 기록하는 모습

이를 build하기 위해 CMakeLists.txt 파일을 적절히 수정하였으며 VSCode를 통해 build하여 executable이 정상적으로 생성되는 것을 확인하였다.

```
#for Problem4
add_library(FSM_OWN STATIC include/FiniteStateMachine/fsm_own.cpp)
target_link_libraries(FSM_OWN EE405_SI_P1)

add_executable(system_integration_own src/system_integration_own.cpp)
target_link_libraries(system_integration_own Threads::Threads EE405_SI_P1 FSM_OWN dxl_sbc_cpp)
```

그림 39. CMakeLists.txt파일을 problem 6D를 위해 수정한 모습

Results

위의 methodology를 이용해 구현한 executable을 실행해 보았으며 terminal 창에 각 motor의 현재 torque 값과 누적된 torque 값이 정상적으로 표시되는 것을 확인할 수 있었다. 여기에 robot에 외부 힘을 주어 motor에 걸리는 부하가 threshold 이상이 되도록 해보았으며 이때 led가 정상적으로 점등이 되며 안전상 주의를 요할 수 있도록 하였고 motor에 걸리는 부하가 줄어들었을 때 다시 led가 소등되면서 계획했던 problem을 효과적으로 해결할 수 있다는 것을 확인하였다. 이러한 방식을 통해 산업현장에서 사용되는 pick-and-place robot을 더욱 안전한 상황에서 사용하고 유지 보수에 도움을 줄 수 있는 자동 기록 system을 구축할 수 있다고 생각한다.

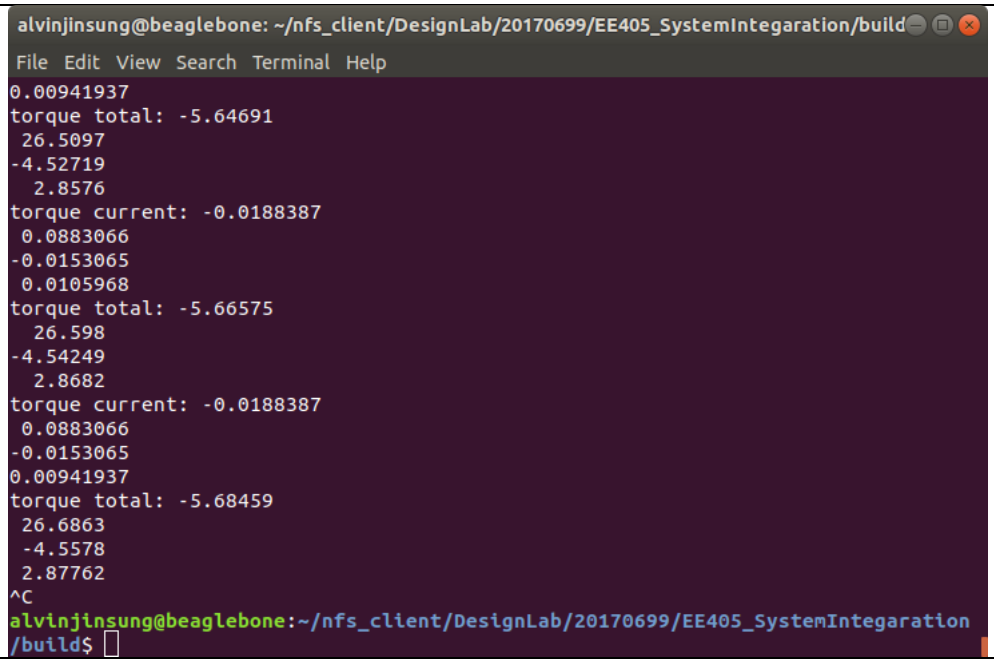


그림 40. Terminal 창에 motor의 현재 torque 값과 누적 torque 값이 표시되는 모습

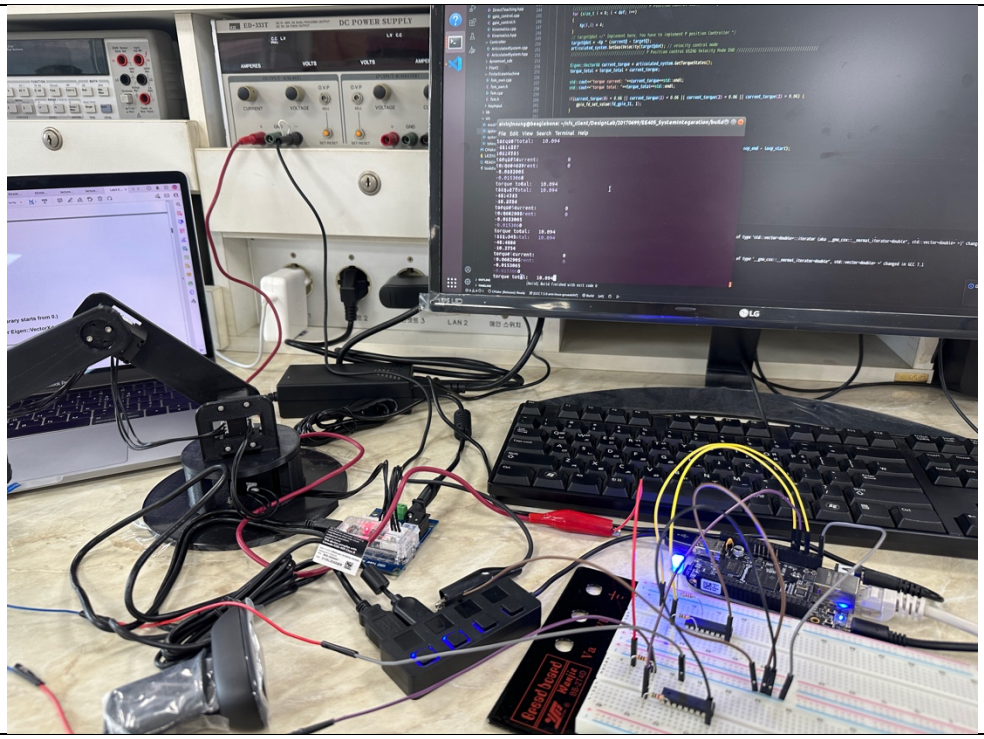


그림 41. Motor에 강한 부하가 걸리자 led가 점등되는 모습

4. Discussion

(1) Are there any examples of finite state machines being used?

Finite state machine의 예시로 vending machine(자판기)를 들 수 있다. 간단한 vending machine FSM을 설계하면 아래와 같이 나타낼 수 있다.

States:
- Idle: default state(no coin inserted)
- SelectProduct: vending machine displays available product to dispense with the total money inserted
- DispenseProduct: dispense the selected product and deduct the appropriate amount of money from the total money

Transitions:
1. Idle -> SelectProduct: inserting coin
2. SelectProduct -> SelectProduct: inserting coin
3. SelectProduct -> DispenseProduct: selecting product
4. DispenseProduct -> SelectProduct: dispensing product
5. DispenseProduct -> Idle: total money all spent

(2) To define state transitions, it was necessary to determine under what conditions they occur and what initialization process is required for the next task. Discuss what initialization process is needed for each state of FSM defined in I.overview.

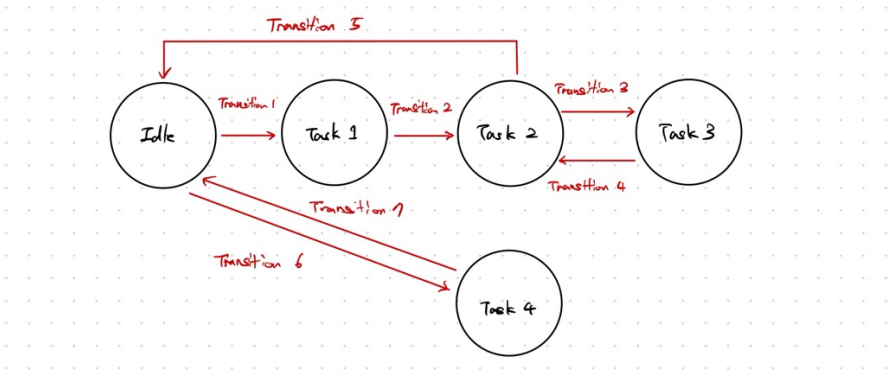
- Idle:
Setting targetQ as current position value

- Task1:
current position and target position value for direct teaching trajectory generation
defining error variable between current position and target position for error norm calculation

- Task2:
GPIO port open for end effector(magnet) to properly work based on the commands
current position value for teleoperation
defining appropriate action for each keyboard input(ex. Movement information(direction, magnitude) for directional keyboard commands)

- Task3:
current position and target position value for direct teaching trajectory generation
defining error variable between current position and target position for error norm calculation

(3) Define and discuss a new FSM that extends the one defined in the I.overview by adding additional states to perform more diverse tasks.



Finite State machine diagram

States :
- Idle : The robot maintains its current position.
- Task 1 : Using a direct teaching method, move the end-effector of the robot to near the target object.
- Task 2 : Using the keyboard input from Ubuntu PC, control the end-effector position of the robot and turn on/off the solenoid magnet to pick or place a target object. This process must use visual feedback through a camera.
- Task 3 : Using a direct teaching method, move the end-effector to the position where the object is to be placed.
- Task 4 : Using a direct teaching method, move the end-effector to the position where human can inspect the manipulator for repair, regular check, etc.

Transition :
- Transition 1 : When the keyboard input is ‘t’.
- Transition 2 : The error norm between q and last way point is less than a certain threshold.
- Transition 3 : When the keyboard input is ‘e’. (When object pick is success, the user gives the input ‘e’)
- Transition 4 : The error norm between q and last way point is less than a certain threshold.
- Transition 5 : When the keyboard input is ‘x’. (When object place is success, the user gives the input ‘x’)
- Transition 6 : When the keyboard input is ‘y’. (When inspection needed, the user gives the input ‘y’)
- Transition 7 : When the keyboard input is ‘z’. (When inspection finished, the user gives the input ‘z’)

(4) FSM can be used for various robotic applications. Walking hopper can be one of these examples. For the task that the walking hopper walks on, how can states and transitions be defined?

States :
- Stand: The hopper is in a standing position.
- PrepareForHop: The hopper is preparing to perform a hop.
- Hop: The hopper is executing a hop.
- Land: The hopper has completed a hop and is landing.
- Rest: The hopper is in a resting state.

Transition :
- Stand -> PrepareForHop: Initiates the hop process by triggering the hopper to start preparing for a hop.
- PrepareForHop -> Hop: Performs the actual hop action, using appropriate mechanisms or actuators to generate upward thrust.
- Hop -> Land: Indicates that the hopper has completed the hop and is transitioning to the landing phase.
- Land -> PrepareForHop: Landing successful and prepare for next hop
- Land -> Rest: Triggers any command to make the hopper be in a resting state after completing the hop or any other task.
- Rest -> Stand: Resets the hopper to its initial state, preparing it for a new hopping cycle.

(5) As mentioned in Problem 6.D, choose your own problem. The report should include problem definition, methodology and result.

Pick-and-place robot의 과도한 부하 또는 외부 힘이 가해질 때 알림 system 및 motor 교체 시기 파악 이라는 problem을 성공적으로 정의하고 methodology를 구축하여 구현 및 실행, result 역시 성공적으로 얻어내었다. 보고서에 해당 부분을 서술하였으며 이를 통해 산업 현장에서의 안전 환경을 향상시킬 수 있을 것이라 생각한다.

5. References

[1] Lab6 Experiment Guide_revised_v3.pdf
[2] Lab1 Procedure.pdf
[3] Lab1 Experiment Guide.pdf
[4] Lab2 Experiment Guide.pdf
[5] Lab2 Procedure.pdf
[6] Lab3 Experiment Guide.pdf
[7] Lab3 Lab Procedures_v02.pdf
[8] Lab4 Experiment Guide.pdf
[9] EE405 Electronic Design Lab4_revised_final.pdf
[10] EE405 Lab5 Experiment Guide.pdf
[11] EE405 Lab5 procedure V2.pdf
[12] lecture_control_part1.pdf
[13] lecture_control_part2.pdf
[14] lecture_robotics_part1.pdf
[15] lecture_robotics_part2.pdf