Robot Arm

W2022

Introduction

My name is Jiwon. I was the main developer for the robot arm during the Winter 2022 term. Feel free to contact me with any questions at jiwon.ha@uwaterloo.ca or message me on Teams.

The robot arm is a part of the industrial automation project being developed in partnership with Brock Solutions. The general goal is to have an arm that is able to communicate with the PLCs on conveyors to pick up and drop off boxes at conveyor platforms. The arm is programmed using C. The microcontroller used for the arm is STM32 Nucleo-F401RE. There are three X-Nucleo-IHM02A1 motor driver boards stacked on top of the microcontroller.

First steps

The very first thing you should do is figure out how to compile and download code onto the Nucleo board. To do this, you will need a software called Keil uVision. Everything is set up on a computer in the second floor Ideas Clinic (ENGIDEAS7 - the computer farthest away from the doors). You can also download it onto your own laptop. The software requires Windows.

On your computer:

- Download the current code from my GitHub: https://github.com/sparadicular/robot-arm-W2022/tree/master
- Navigate to robotic_arm > Projects > MDK-ARM
- Open the uVision project named Project.
- Go to the top and select Project > Rebuild all target files. Wait for the build to finish.
- Connect the Nucleo board to the computer via USB. Go to the top and select Flash > Download.
- You will see the LED on the Nucleo board flash red and green repeatedly.

Once you are finished, find the robot arm wiring document on Teams and make sure the board is wired properly. You will need to familiarize yourself with the hardware of the arm as well as how to look at diagrams and read documentation.

Once you have the wiring done, find the main program in the files and try running just the homing sequence. You will need to comment out everything after the homing sequence in main. The arm should home so that all joints are at 90 degrees.

Development during W2022

Some components I worked on during my term.

Inverse and Forward Kinematics

In order to achieve the goal of the robot arm being able to pick up and drop off from conveyors, it must be able to go to a specified location and orientation. Simply put, forward kinematics finds the location of the end effector (gripper) of the robot arm from given joint angles. Inverse kinematics does the opposite; it finds the joint angles from a given end effector position and orientation.

There is a useful book that you can reference in the co-op office.

The functions in the main.c file use inverse kinematics equations. There is a version using equations generated using vector and matrix algebra, and there is another using equations obtained from trig. Currently, the trig version is being used. However, it only has the base, shoulder, and elbow angles. The equations for the wrist angles need to be added.

The matrix algebra version has all the equations, but it needs to be refined (take into consideration the signs of the trig in the equations so that the correct angle is produced). You can take the wrist angle equations from here and add it to the trig equations.

UART Communication

The robot arm must be able to receive data from the conveyors on which locations to pick up from and drop off to. This is currently done through serial communication with the PLCs. Serial communication on the microcontroller is done through UART.

Look through the code and see how UART is set up:
In the beginning of the main.c file, you can see the declaration for UART2:

```
UART_HandleTypeDef huart2;
static void MX_USART2_UART_Init(void);
```

The initialization function that comes later looks like this:

```
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

UART 2 is connected to the microcontroller's serial cable (the same cable used for flashing).

The serial connection to the PLCs is currently configured as UART 6 in the code. You must wire the USB connector according to the wiring diagram for UART 6 to communicate through this channel.

This is a function that receives data from serial:

```
HAL_UART_Receive(&huart2, hold_message, 4, HAL_MAX_DELAY);
```

The above function call receives 4 bytes of data from UART 2 into a string called hold_message, and waits for up to HAL_MAX_DELAY amount of time for the bytes to be received.

There is a similar function to transmit data through UART.

Hardware Modifications

The motor on the gripper was originally a DC motor. This was replaced with a servo motor and an entirely new gripper. The code for the servo currently has two issues: 1) it has not been integrated into the main arm code. It is in a separate program that I used to test just the servo. 2) The servo did not run properly with the servo code. It makes clicking noises, but does not move.
The gripper has a few issues of its own. There is more on the gripper in a separate document.

There is a limit switch that needs to be mechanically attached to the arm for the homing of the gripper. The limit switch has already been wired to the microcontroller and is functioning.

Note: do not run the servo off the Nucleo board. It could kill the board by drawing too much current. Use one of the variable power supplies inside the large grey storage unit in the second floor Ideas Clinic.

Linear Interpolation

There is a linear interpolation function in the main.c file. This function takes in a target point and creates points on a linear path from the current point and the target point, and then moves through those points. More specifically:

1. Subtracts the current point from the target point (effectively creating a vector)
2. Calculates the magnitude of this vector
3. Gets the number of points on the path by dividing the magnitude by a number representing the distance between the points on the linear path
4. Divides the vector from 1 by the resulting number from 3
5. Adds the resulting vector from 4 to the current point in order to get the next point on the path. This is done repeatedly until the target point is reached.

Trajectory Planning

This function takes in a number of points (each represented by the position and orientation of the end effector) and moves through those points.

Note: Currently, I have a sort of user interface set-up in this function, which was for my own purposes. Feel free to remove it. Stuff that looks like this:

```
USART_Transmit(&huart2, "For each point, enter the position and end effector vectors in the following format.\n");
USART_Transmit(&huart2, "For each point, enter the position and end effector vectors in the following format.\n");
USART_Transmit(&huart2, "px py px\n");
USART_Transmit(&huart2, "ux uy uz\n");
USART_Transmit(&huart2, "vx vy vz\n");
USART_Transmit(&huart2, "wx wy wz\n");
USART_Transmit(&huart2, "The format must be: xxx(space)yyy(space)zzz(enter). If a number is less than 3 digits, fill remaining digits with spaces.");
USART_Transmit(&huart2, "To restart your current vector, enter 777 for x,y,or z.\n");
USART_Transmit(&huart2, "To restart your previous vector, enter 888 for x,y,or z, repeatedly to move back several vectors.\n");
USART_Transmit(&huart2, "To restart your entire entry, enter 999 for x,y,or z.\n");
```

You can use it if it's helpful to you, but it may just be confusing. You can remove all the rest of the code with the for loops and if statements dealing with user input. I was inputting points on a serial monitor for testing purposes. The arm will not be taking user input in its final form. It will take input from the PLCs.

Note: both the linear path and trajectory planning function do not follow a smooth path since the number of points is small. As the number of points increases, the path should get smoother.

Current Issues/Next Steps

- The status registers in the motor drivers do not work. All values are always read as zero. There is no way to tell the busy status of a single motor. This is a problem because each command to a motor must wait for the previous command to be finished, and this is done by reading the busy bit in the status register. The current code compromises by reading the busy pin instead, but the busy pin is shared amongst all the motors, so the motors can only move one at a time. This creates very slow and unsmooth movements. (Not sure if there is a solution to this)
- The base does not move in the linear path function. This needs to be debugged.
- The trajectory function in general needs a lot of fixing
- Figure out what is causing the servo to click and not move
- Integrate servo code into main file
- Explore velocity and acceleration for the linear path function and trajectory planning (ask Dr. Bedi)