Introduction

The eebot term project's concept is to develop an assembly project that assembles a robot equipped with an HCS12 microcontroller and guides it from the Start to Finish gates. The most efficient way to complete this project is by making left turns when faced with an intersection, as this algorithm is guaranteed to get the bot to the Finish. In the rest of this report, there will be explanations of specific key sections of our project code, followed by a few problems that were encountered and some key takeaways.

Code Explanation

```
*****************************************************************
* Demonstration Program *                                       *
* Authors: Krish Patel, Simrat Gill, Sanjay Sivapragasm *       *
* eeBOT 31445                                                   *
* Tuesday November 28, 2023 10:14am                             *
* Section #4 - TA: Mohsen Ensafjoo                              *
*****************************************************************
```

The code block to the right is the initial start-up to initialize the key subroutines and components, such as the motors, the LCD, timer overflow, and the messages to be printed on the LCD. Then, the MAIN section runs through a series of subroutines that display the sensor values.

```
            ORG     $4000
;           Initialization
Entry:
_Startup:
            LDS     #$4000
            CLI
            JSR     INIT
            JSR     openADC
            JSR     initLCD
            JSR     CLR_LCD_BUF
            BSET    DDRA,%00000011
            BSET    DDRT,%00110000
            JSR     initAD
            JSR     initLCD
            JSR     clrLCD
            LDX     #msg1
            JSR     putsLCD
            LDAA    #$C0
            JSR     cmd2LCD
            LDX     #msg2
            JSR     putsLCD
            JSR     ENABLE_TOF

;           Display Sensors
MAIN
            JSR     G_LEDS_ON
            JSR     READ_SENSORS
            JSR     G_LEDS_OFF
            JSR     UPDT_DISPL
            LDAA    CRNT_STATE
            JSR     DISPATCHER
            BRA     MAIN
```

Figure 1: Initial startup code

```
;*************************************
; Sub-Routine Section
;*************************************

DISPATCHER      JSR     VERIFY_START
                RTS

VERIFY_START    CMPA    #START
                BNE     VERIFY_FWD
                JSR     START_ST
                RTS

VERIFY_FWD      CMPA    #FWD
                BNE     VERIFY_STP
                JSR     FWD_ST
                RTS

VERIFY_REV      CMPA    #REV
                BNE     VERIFY_L_ALGN
                JSR     REV_ST
                RTS

VERIFY_STP      CMPA    #ALL_STP
                BNE     VERIFY_L_TRN
                JSR     ALL_STOP_ST
                RTS

VERIFY_L_TRN    CMPA    #L_TRN
                BNE     VERIFY_R_TRN
                JSR     LEFT
                RTS

VERIFY_L_ALGN   CMPA    #L_ALGN
                BNE     VERIFY_R_ALGN
                JSR     L_ALGN_DONE
                RTS

VERIFY_R_TRN    CMPA    #R_TRN
                BNE     VERIFY_REV
                JSR     RIGHT

VERIFY_R_ALGN   CMPA    #R_ALGN
                JSR     R_ALGN_DONE
                RTS
```

The next main section of the project code is the sub-routine section. This section outlines the main state machine used for the eebot's maze navigation. Following the states tells the eebot what instructions to follow, thus accurately following the black tape in the maze. During this process, each state needs to be verified, and upon verification, the eebot can proceed with the current state or next state. For instance, when verifying the START state, the sub-routine will branch to the FWD state if it is not equal to the START state; otherwise, the START state can be confirmed. This process is repeated throughout the rest of the states until the final state: R_ALGN. If it cannot be verified at this state, it is simply an invalid state, as there are no other possible states available.

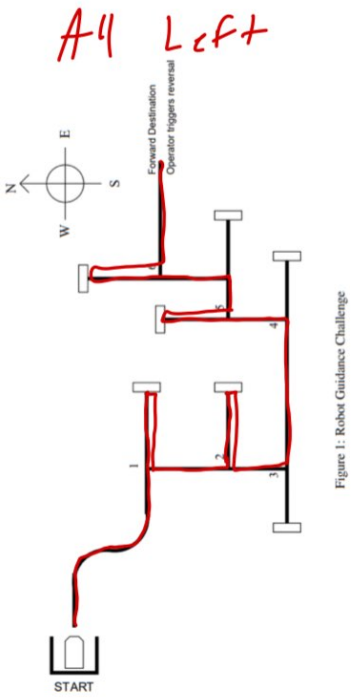

Figure 2: Path the bot followed and the code that was used to do it

Problems Encountered

1. The first significant issue encountered was the bot was not making any adjustments to its path or performing any turns. The first step to troubleshooting was to check the states to see if anything was missing, but after further inspection of the entire project from the top it was noticed that there was a missing “RTS” at the end of the DISPATCHER subroutine.

```
DISPATCHER      JSR     VERIFY_START
                RTS
```

2. The screenshot above shows that the value 6000 was loaded into accumulator Y. This value is used to determine the delay, as it would be multiplied by 50 µs. Initially, the sample value was 600 instead of 6000, which led to issues with the eebot following the black tape correctly.

LDY #6000	LDY #600
$6000 \times 50 \times 10^{-6} = 0.3$ sec	$600 \times 50 \times 10^{-6} = 0.03$ sec

As the calculations above indicate, a larger value used would result in a longer delay, whereas a smaller value used with a shorter delay. Using 6000 allowed the eebot to have a longer delay and, therefore, more accurately followed the black tape as it had more time to determine the difference between black versus white. When using 600 initially, this led to our bot checking more rapidly due to smaller increments caused by the shorter delay. As a result, encountered issues during the maze when turning left and when turning around after performing a 180-degree turn when reaching a wall in the maze. Overall, this section of the code was not too difficult to debug, but it is important to understand the purpose of the delay and how it can impact the bot's performance.

```
PARTIAL_L_TRN   LDY     #6000
                JSR     del_50us
                JSR     INIT_LEFT
                MOVB    #L_TRN, CRNT_STATE
                LDY     #6000
                JSR     del_50us
                BRA     EXIT
```

3. Another issue that was faced was determining the adequate threshold values that were required for the guider code portion of the eebot. After performing a lot of troubleshooting for the rest of the project code, issues with the eebot movement were still present. The measured threshold values using a sample intersection created by black tape on a piece of paper to mirror the actual maze. Despite taking accurate measurements, the eebot movement issues did not resolve. It primarily was moving forward and turned at the wrong time, or did not recognize a turn. After troubleshooting the main issue with our approach. The exact values that were measured and displayed in the LCD of the eebot were selected as the threshold values in the code. This was too small of a margin when navigating the maze, as the bot was unable to determine the difference between black and white. A margin was established to rectify this by subtracting ten from the measured values. The reason behind the margin is to allow the bot to detect changes and determine orientation. This change resolved issues with navigating the maze, as the bot could now accurately determine the difference between black and white. Although it was a minor change, it took a longer time to debug and resolve. To avoid such an error in the future, it is

essential to gain a better grasp and understanding of the purpose of the guider code and how it functions in relation to the eebot sensors and threshold values.

```
BASE_LINE           FCB $9D
BASE_BOW            FCB $CA
BASE_MID            FCB $CA
BASE_PORT           FCB $CC
BASE_STBD           FCB $CC
```

Key Takeaways

The main takeaway from this project is that when building such a large project, even the smallest of instructions can make the difference between a working project and one that makes mistakes, as is evident in the problems section, especially the first problem encountered regarding the single missing ‘RTS’ instruction. Another takeaway is that it is easy to lose track of what has been done and what is still remaining to be done, therefore, it is important to have a well-thought-out plan in advance to progress at a good pace. Lastly, it is crucial to understand the workings of the eebot as well as the use of the various instructions and programs that will be implemented. If a part cannot be understood, getting help from the TA is always reasonable, and it can make the difference between success and failure.