# COE538 - Robot Guidance Challenge

Final Report

Professor: Dr. Vadim Geurkov

Section Number: 11

Developed by:

Andrew Le - 501090188

Taha Ghori - 501102847

Zoravar Multani - 501110311

Code Explanations:

```
;*********************************
;* Main Code
;*********************************
                ORG   $4000
Entry:
_Startup:
                LDS   #$4000
                CLI


                JSR   initPORTS
                JSR   initAD
                JSR   initLCD
                JSR   clrLCD
                JSR   initTCNT


                JSR CLR_LCD_BUF
                JSR CLR_LCD_BUF
                LDX #msg1
                JSR putsLCD
                LDAA #$C0
                JSR cmd2LCD
                LDX #msg2
                JSR putsLCD

;*********************************
;* Main Loop
;*********************************
MAIN:           JSR   UPDT_READING
                JSR   UPDT_DISPL
                LDAA  CRNT_STATE
                JSR   DISPATCHER
                BRA   MAIN
```

This comprises the startup and main code segments. The startup section initializes various functions essential for the eebot. Meanwhile, the main code continuously executes four lines in repetition to run specific subsections of the code.

```
;*********************************
;* EEBot Subroutines (Dispatcher)
;*********************************

;-------------------------------
DISPATCHER      CMPA  #START
                BNE   NOT_STRT
                JSR   START_ST
                RTS
;-------------------------------
NOT_STRT        CMPA  #FWD
                BNE   NOT_FWD
                JMP   FWD_ST
                RTS
;-------------------------------
NOT_FWD         CMPA  #RT_TRN
                BNE   NOT_RT_TRN
                LDY   #1000
                JSR   del_50us
                JSR   RT_TRN_ST
                RTS
;-------------------------------
NOT_RT_TRN      CMPA  #LFT_TRN
                BNE   NOT_LFT_TRN
                LDY   #1000
                JSR   del_50us
                JSR   LFT_TRN_ST
                RTS
;-------------------------------
NOT_LFT_TRN     CMPA  #REV
                BNE   NOT_REV
                JSR   REV_ST
                RTS
;-------------------------------
NOT_REV         CMPA  #BACK_TRCK
                BNE   NOT_BACK_TRCK
                JMP   BACK_TRCK_ST
                RTS
;-------------------------------
NOT_BACK_TRCK   CMPA  #STND_BY
                BNE   NOT_STND_BY
                JSR   STND_BY_ST
                RTS
;-------------------------------
NOT_STND_BY     NOP
DISP_EXIT       RTS
```

This code serves as the Dispatcher, aiding the eebot in determining the subsequent process to execute. The accompanying code instructs the eebot to track the alternating black line by transitioning between various rotation states.

1

```
;*****************************************
;* EEBot State Machine
;*****************************************

START_ST        BRCLR PORTAD0,$04,NO_FWD
                JSR   INIT_FWD
                MOVB  #FWD,CRNT_STATE
                BRA   START_EXIT
NO_FWD          NOP

START_EXIT      RTS
;-----------------------------------------
FWD_ST          PULD

                BRSET PORTAD0,$04,NO_FWD_BUMP

                LDAA  SEC_PATH
                STAA  NEXT_DIR
                JSR   INIT_REV
                MOVB  #REV,CRNT_STATE
                JMP   FWD_EXIT

NO_FWD_BUMP     BRSET PORTAD0,$08,NO_REAR_BUMP
                JMP   INIT_STND_BY
                MOVB  #STND_BY,CRNT_STATE
                JMP   FWD_EXIT


NO_REAR_BUMP    LDAA  D_DETN
                BEQ   NO_D_DETECT

                JSR   INIT_FWD
                MOVB  #FWD, CRNT_STATE

                LDY   #570
                JSR   del_50us

                JSR   INIT_RT_TRN
                MOVB  #RT_TRN,CRNT_STATE
                JMP   FWD_EXIT

NO_D_DETECT     LDAA B_DETN
                BEQ NO_B_DETECT
                LDAA A_DETN
                BEQ LFT_TURN

                BRA NO_SHFT_LT
```

2

```
LFT_TURN        JSR   INIT_FWD
                MOVB  #FWD, CRNT_STATE
                LDY   #570
                JSR   del_50us

                JSR   INIT_LFT_TRN
                MOVB  #LFT_TRN,CRNT_STATE
                JMP   FWD_EXIT

NO_B_DETECT     LDAA  F_DETN
                BEQ   NO_SHFT_LT
                JSR   PORTON

RT_FWD_DIS      LDD   COUNT2
                CPD   #INC_D
                BLO   RT_FWD_DIS
                JSR   INIT_FWD
                JMP   FWD_EXIT

NO_SHFT_RT      LDAA  E_DETN
                BEQ   NO_SHFT_RT
                JSR   STARON

LT_FWD_DIS      LDD   COUNT1
                CPD   #INC_D
                BLO   LT_FWD_DIS
                JSR   INIT_FWD
                JMP   FWD_EXIT

NO_SHFT_LT     JSR   STARON
                JSR   PORTON

FWD_STR_DIS     LDD   COUNT1
                CPD   #FWD_D
                BLO   FWD_STR_DIS
                JSR   INIT_FWD

FWD_EXIT        JMP   MAIN

;-----------------------------------------
REV_ST          LDD   COUNT1
                CPD   #REV_D
                BLO   REV_ST
                JSR   STARFWD
                LDD   #0
                STD   COUNT1
```

3

```
REV_U_TRN       LDD   COUNT1
                CPD   #UTRN_D
                BLO   REV_U_TRN
                JSR   INIT_FWD
                LDAA  RETURN
                BNE   BACK_TRCK_REV
                MOVB  #FWD,CRNT_STATE
                BRA   REV_EXIT

BACK_TRCK_REV     JSR   INIT_FWD
                MOVB  #BACK_TRCK,CRNT_STATE

REV_EXIT        RTS

;-----------------------------------------
RT_TRN_ST       LDD   COUNT2
                CPD   #STR_D
                BLO   RT_TRN_ST
                JSR   STAROFF
                LDD   #0
                STD   COUNT2

RT_TURN_DEL     LDD   COUNT2
                CPD   #TRN_D
                BLO   RT_TURN_DEL
                JSR   INIT_FWD
                LDAA  RETURN
                BNE   BACK_TRCK_RT_TRN
                MOVB  #FWD,CRNT_STATE
                BRA   RT_TRN_EXIT

BACK_TRCK_RT_TRN  MOVB  #BACK_TRCK,CRNT_STATE

RT_TRN_EXIT     RTS

;-----------------------------------------
LFT_TRN_ST       LDD   COUNT1
                CPD   #STR_D
                BLO   LFT_TRN_ST
                JSR   PORTOFF
                LDD   #0
                STD   COUNT1

LFT_TURN_DEL     LDD   COUNT1
                CPD   #TRN_D
                BLO   LFT_TURN_DEL
                JSR   INIT_FWD
                LDAA  RETURN
                BNE   BACK_TRCK_LFT_TRN
                MOVB  #FWD,CRNT_STATE
                BRA   LFT_TRN_EXIT
```

4

```
BACK_TRCK_LFT_TRN  MOVB  #BACK_TRCK,CRNT_STATE

LFT_TRN_EXIT    RTS

;-----------------------------------------
BACK_TRCK_ST      PULD
                BRSET PORTAD0,$08,NO_BK_BUMP
                JSR   INIT_STND_BY
                MOVB  #STND_BY,CRNT_STATE
                JMP   BACK_TRCK_EXIT

NO_BK_BUMP      LDAA  NEXT_DIR
                BEQ   GOOD_PATH
                BNE   BAD_PATH

;-----------------------------------------
GOOD_PATH       LDAA  D_DETN
                BEQ   NO_RT_TRN

                PULA
                PULA
                STAA  NEXT_DIR
                JSR   INIT_RT_TRN
                MOVB  #RT_TRN,CRNT_STATE
                JMP   BACK_TRCK_EXIT

NO_RT_TRN       LDAA  B_DETN
                BEQ   RT_LINE_S
                LDAA  A_DETN
                BEQ   LEFT_TURN
                PULA
                PULA
                STAA  NEXT_DIR
                BRA   NO_LINE_S

LEFT_TURN       PULA
                PULA
                STAA  NEXT_DIR
                JSR   INIT_LFT_TRN
                MOVB  #LFT_TRN,CRNT_STATE
                JMP   BACK_TRCK_EXIT

;-----------------------------------------
BAD_PATH        LDAA  B_DETN
                BEQ   NO_LFT_TRN
                PULA
                STAA  NEXT_DIR
                JSR   INIT_LFT_TRN
                MOVB  #LFT_TRN,CRNT_STATE
                JMP   BACK_TRCK_EXIT
```

```
NO_LFT_TRN      LDAA  D_DETN
                BEQ   RT_LINE_S
                LDAA  A_DETN
                BEQ   RIGHT_TURN
                PULA
                STAA  NEXT_DIR
                BRA   NO_LINE_S

RIGHT_TURN      PULA
                STAA  NEXT_DIR
                JSR   INIT_RT_TRN
                MOVB  #RT_TRN,CRNT_STATE
                JMP   BACK_TRCK_EXIT

;---------------------------------------
RT_LINE_S       LDAA  F_DETN
                BEQ   LT_LINE_S
                JSR   PORTON

RT_FWD_D        LDD   COUNT2
                CPD   #INC_D
                BLO   RT_FWD_D
                JSR   INIT_FWD
                JMP   BACK_TRCK_EXIT

LT_LINE_S       LDAA  E_DETN
                BEQ   NO_LINE_S
                JSR   STARON

LT_FWD_D        LDD   COUNT1
                CPD   #INC_D
                BLO   LT_FWD_D
                JSR   INIT_FWD
                JMP   BACK_TRCK_EXIT

NO_LINE_S       JSR   STARON
                JSR   PORTON

FWD_STR_D:      LDD   COUNT1
                CPD   #FWD_D
                BLO   FWD_STR_D
                JSR   INIT_FWD

BACK_TRCK_EXIT  JMP   MAIN
```

```
STND_BY_ST      BRSET PORTAD0,$04,NO_START
                BCLR  PTT,%00110000
                MOVB  #START,CRNT_STATE
                BRA   STND_BY_EXIT

NO_START        NOP

STND_BY_EXIT    RTS
```

All the code after the Dispatcher is part of the state machine. This state machine contains the entire logic of the eebot completing the maze by itself. By using the provided Guider code and the state machine + motor code from lab5, we were able to simplify the design of project into just making a more complex state machine which was able to solve the maze. One of the subroutines updates the dispatcher based on the information provided by the LEDs and photoresistors on the bottom of the eebot.

**Problems Encountered:**

- Robot would start maze, enter the beginning curve, then turn and around and exit maze
    - Accidentally deleted portion of code without realizing. Realization of importance of saving backups and version control. Had to redo work from a very old version.
- Robot made 2 360 degree turns when detecting any intersection and then continued
    - Accidentally included logic for left side of bot line detection when the robot is based on only right turn logic, this caused the dispatcher and FSM to be stuck in loop.
- Robot would sometimes drive off the line and not correct itself
    - Forgot to include condition in where the front sensors were not on the line, simple fix involving their detection.

There were also some other problems, but they were quite small and easy fixes so it is hard to remember and not worth noting. We were able to solve all the major problems we had. One of our key take aways is using GIT version control for any large software project since it really saves quite a lot of time and energy.

**Conclusion:**

Overall, this was an incredibly insightful project that we are deeply proud of. It helped to really form a deeper understanding of how code and hardware interact and gave us a new found appreciation for higher level programming languages such as C and Python.

We also realized the importance of version control and will be implementing it for any future projects that we work on. By using convenient services such as GitHub, we can easily do this with little to no extra hassle.

By using the age-old logic of "stick to the right wall" we were able to solve the maze through only making right turns. Although this does not result in the fastest time, this logic provides a guaranteed way to solve any labyrinth.