**Part 1**

**Problem 18.6**: Bode Plots

1. Bode Plot for Earthquake Excitation

1e-06 s^3 + 0.001824 s^2 + 1.071 s + 201.4

TA(s) = ---------------------------------------------------------------------------------------------

s^6 + 0.05 s^5 + 32.36 s^4 + 0.7688 s^3 + 237.9 s^2 + 1.071 s + 201.4



1. Bode Plot for Wind Excitation

0.001 s^4 + 4e-05 s^3 + 0.02432 s^2 + 0.0003648 s + 0.1071

TB(s) = ----------------------------------------------------------------------------------------------

s^6 + 0.05 s^5 + 32.36 s^4 + 0.7688 s^3 + 237.9 s^2 + 1.071 s + 201.4



**Problem 18.11**

|  |  |
| --- | --- |
| C:\Users\pthb3\AppData\Local\Temp\ConnectorClipboard6555706118390356564\image15107739386890.png | C:\Users\pthb3\AppData\Local\Temp\ConnectorClipboard6555706118390356564\image15107741416980.png |
| C:\Users\pthb3\AppData\Local\Temp\ConnectorClipboard6555706118390356564\image15107741669210.png | C:\Users\pthb3\AppData\Local\Temp\ConnectorClipboard6555706118390356564\image15107741836480.png |
| C:\Users\pthb3\AppData\Local\Temp\ConnectorClipboard6555706118390356564\image15107741968500.png | C:\Users\pthb3\AppData\Local\Temp\ConnectorClipboard6555706118390356564\image15107742120530.png |

**Part 3**

**Code**: hw1\_config.h

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* hw1\_config.h  \*  \* Contains the settings for configuration of hw1.c  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #ifndef HW1\_CONFIG  #define HW1\_CONFIG  // Set constants  #define GEAR\_RATIO 35.555 // Motor gear ratio  #define ENCODER\_RES 60 // Encoder resolution  #define MOTOR\_CHANNEL\_L 3 // Left motor channel  #define MOTOR\_CHANNEL\_R 2 // Right motor channel  #define MOTOR\_POLARITY\_L 1 // Left motor polarity  #define MOTOR\_POLARITY\_R -1 // Right motor polarity  #define ENCODER\_CHANNEL\_L 3 // Left encoder channel  #define ENCODER\_CHANNEL\_R 2 // Right encoder channel  #define ENCODER\_POLARITY\_L 1 // Left encoder polarity  #define ENCODER\_POLARITY\_R -1 // Right encoder polarity  #define D\_GAIN 0.9 // Proportional loop gain  #define SETPOINT\_ZERO 0 // Zero set point for stationary wheel  #define SAMPLE\_RATE\_HZ 100 // Loop rate  #define DT 0.01 // 1/SAMPLE\_RATE\_HZ  #endif //HW1\_CONFIG |

**Code**: hw1.c

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* File: hw1.c  \* Author: Parker Brown  \* Date: 11/15/2017  \* Course: MAE 144, Fall 2017  \* Description: Program runs closed feedback loop, minimizing angle error between  \* left and right wheel of MIP. User spins right wheel, while left wheel tracks  \* right wheel angle.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // usefulincludes is a collection of common system includes for the lazy  // This is not necessary for roboticscape projects but here for convenience  // Nice to have for TWO\_PI  #include <rc\_usefulincludes.h>  // main roboticscape API header  #include <roboticscape.h>  #include "hw1\_config.h"  // function declarations  void on\_pause\_pressed(); // do stuff when paused button is pressed  void on\_pause\_released(); // do stuff when paused button is released  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* int main()  \*  \* hw1 main function contains these critical components  \* - call to rc\_initialize() at the beginning  \* - main while loop that checks for EXITING condition  \* - Run feedback loop while RUNNING or wait while PAUSED  \* - rc\_cleanup() at the end  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int main(){  // initialize hardware first  if(rc\_initialize()){  fprintf(stderr,"ERROR: failed to initialize rc\_initialize(), are you root?\n");  return -1;  }  // initialize stuff here  rc\_set\_pause\_pressed\_func(&on\_pause\_pressed);  rc\_set\_pause\_released\_func(&on\_pause\_released);  rc\_enable\_motors();  // done initializing so set state to RUNNING  rc\_set\_state(RUNNING);  // Initialize variables used in the while loop  int sleep\_time=DT\*1e6; // Sleep time to set rough loop rate  float wheelAngleL=0, wheelAngleR=0; // Initialize wheel angles to zero  float dutyL=0; // Initialize duty cycle to zero  float errorL=0; // Initialize loop error to zero  // Print header for standard output  printf("Loop Gain: %3.1f\n", D\_GAIN);  printf("Wheel Angle Phi (Rad)\n");  printf(" Phi\_L |");  printf(" Phi\_R |");  printf(" Error\_L |");  printf(" Duty\_L |");  printf(" \n");  // Keep looping until state changes to EXITING  while(rc\_get\_state()!=EXITING){  // If RUNNING, run feedback loop  if(rc\_get\_state()==RUNNING){  rc\_set\_led(GREEN, ON); // GREEN when on  rc\_set\_led(RED, OFF); // RED when paused  // Get wheel angles by reading encoder channels with math  // Math says (wheel angle) = 2pi \* (enc position) / (enc count per rev)  wheelAngleL = ((rc\_get\_encoder\_pos(ENCODER\_CHANNEL\_L) \* TWO\_PI) \  / (ENCODER\_POLARITY\_L \* GEAR\_RATIO \* ENCODER\_RES));  wheelAngleR = ((rc\_get\_encoder\_pos(ENCODER\_CHANNEL\_R) \* TWO\_PI) \  / (ENCODER\_POLARITY\_R \* GEAR\_RATIO \* ENCODER\_RES));  errorL = wheelAngleR - wheelAngleL; // Error between free and driven wheel  // errorL = SETPOINT\_ZERO - wheelAngleL; // Error with zero setpoint  dutyL = D\_GAIN \* errorL; // Controller output == left wheel duty cycle  // Check for motor saturation  if(dutyL > 1.0){  dutyL = 1.0;  }  else if(dutyL < -1.0){  dutyL = -1.0;  }  rc\_set\_motor(MOTOR\_CHANNEL\_L, MOTOR\_POLARITY\_L \* dutyL); // drive left wheel  // Print wheel angles, angle error, and contoller output duty cycle  printf("\r"); // carriage return because it looks pretty  printf("%8.3f |", wheelAngleL);  printf("%8.3f |", wheelAngleR);  printf("%8.3f |", errorL);  printf("%8.3f |", dutyL);  fflush(stdout);  }  else if(rc\_get\_state()==PAUSED){  // Set everything to an off state when paused  rc\_set\_led(GREEN, OFF); // GREEN when on  rc\_set\_led(RED, ON); // RED when paused  rc\_set\_motor\_free\_spin\_all(); // Set motors to free spin while paused  rc\_set\_encoder\_pos(MOTOR\_CHANNEL\_L, 0); // Reset left encoder position  rc\_set\_encoder\_pos(MOTOR\_CHANNEL\_R, 0); // Reset right encoder position  }  usleep(sleep\_time); // Sleep for DT in microseconds  }  // exit cleanly  rc\_cleanup();  return 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* void on\_pause\_released()  \*  \* Make the Pause button toggle between paused and running states.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void on\_pause\_released(){  // toggle betewen paused and running modes  if(rc\_get\_state()==RUNNING) rc\_set\_state(PAUSED);  else if(rc\_get\_state()==PAUSED) rc\_set\_state(RUNNING);  return;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* void on\_pause\_pressed()  \*  \* If the user holds the pause button for 2 seconds, set state to exiting which  \* triggers the rest of the program to exit cleanly.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void on\_pause\_pressed(){  int i=0;  const int samples = 100; // check for release 100 times in this period  const int us\_wait = 2000000; // 2 seconds  // now keep checking to see if the button is still held down  for(i=0;i<samples;i++){  rc\_usleep(us\_wait/samples);  if(rc\_get\_pause\_button() == RELEASED) return;  }  printf("long press detected, shutting down\n");  rc\_set\_state(EXITING);  return;  } |

**Code**: Makefile

|  |
| --- |
| # This is a general use makefile for robotics cape projects written in C.  # Just change the target name to match your main source code filename.  TARGET = hw1  CC := gcc  LINKER := gcc -o  CFLAGS := -c -Wall -g  LFLAGS := -lm -lrt -lpthread -lroboticscape  SOURCES := $(wildcard \*.c)  INCLUDES := $(wildcard \*.h)  OBJECTS := $(SOURCES:$%.c=$%.o)  prefix := /usr/local  RM := rm -f  INSTALL := install -m 4755  INSTALLDIR := install -d -m 755  LINK := ln -s -f  LINKDIR := /etc/roboticscape  LINKNAME := link\_to\_startup\_program  # linking Objects  $(TARGET): $(OBJECTS)  @$(LINKER) $(@) $(OBJECTS) $(LFLAGS)  # compiling command  $(OBJECTS): %.o : %.c $(INCLUDES)  @$(CC) $(CFLAGS) -c $< -o $(@)  @echo "Compiled: "$<  all:  $(TARGET)  debug:  $(MAKE) $(MAKEFILE) DEBUGFLAG="-g -D DEBUG"  @echo " "  @echo "$(TARGET) Make Debug Complete"  @echo " "  install:  @$(MAKE) --no-print-directory  @$(INSTALLDIR) $(DESTDIR)$(prefix)/bin  @$(INSTALL) $(TARGET) $(DESTDIR)$(prefix)/bin  @echo "$(TARGET) Install Complete"  clean:  @$(RM) $(OBJECTS)  @$(RM) $(TARGET)  @echo "$(TARGET) Clean Complete"  uninstall:  @$(RM) $(DESTDIR)$(prefix)/bin/$(TARGET)  @echo "$(TARGET) Uninstall Complete"  runonboot:  @$(MAKE) install --no-print-directory  @$(LINK) $(DESTDIR)$(prefix)/bin/$(TARGET) $(LINKDIR)/$(LINKNAME)  @echo "$(TARGET) Set to Run on Boot" |

**Code**: README.txt

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Files: hw1.c, hw1\_config.h, Makefile  \* Author: Parker Brown  \* Date: 11/15/2017  \* Course: MAE 144, Fall 2017  \* Description: Program runs closed feedback loop, minimizing angle error between  \* left and right wheel of MIP. User spins right wheel, while left wheel tracks  \* right wheel angle.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |