Leonardo Fusser (1946995) January 30th, 2022

Week 1 & 2 HW

Computerized Systems Optimization

Serge Hould

Intro multi-threading HW:

Q1) Console output:

```
In main: creating a thread
Main: returns!
```

Q2) Console output:

```
In main: creating a thread
Hello World! It’s me, thread!
Thread: terminating!
```

Q3) Console output:

```
In main: creating a thread
Hello World! It’s me, thread!
Thread: terminating!
Main: returns!
```

Q4) Console output:

```
In main: creating thread
It’s me, thread_routine1!
It’s me, thread_routine2!
It’s me, thread_routine1!
It’s me, thread_routine2!
Exiting thread_routine1
Exiting main thread
```

Q5) Console output:

```
In main: creating thread
It’s me, thread_routine1!
It’s me, thread_routine2!
It’s me, thread_routine1!
It’s me, thread_routine2!
Exiting thread_routine1
Exiting thread_routine2
Exiting main thread
```

Leonardo Fusser (1946995) January 30th, 2022

Synchronization and Mutex HW:

Q1) Modified code to protect and synchronize data:

```
/* Globals. */
static int cnt = 0;

/* Mutex declaration. */
static pthread_mutex_t mutex_print = PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t mutex_cnt = PTHREAD_MUTEX_INITIALIZER;
...
void* thread_routine1(void* threadid){
        int temp_Thread1CNT = 0;
        while(1){
                temp_Thread1CNT = mutex_GetCNT();
                mutex_print("Hello World! It's me, thread_routine1! Count: %d\n", &temp_Thread1CNT);
                mutex_SetCNT(temp_Thread1CNT++);
        }
        return NULL;
}
...
void* thread_routine2(void* threaded){
        int temp_Thread2CNT = 0;
        while(1){
                temp_Thread2CNT = mutex_GetCNT();
                mutex_print("Hello World! It's me, thread_routine2! Count: %d\n", temp_Thread2CNT);
                mutex_SetCNT(temp_Thread2CNT++);
        }
        return NULL;
}
...
```

```
void mutex_print(char *temp_PrintSTR, int temp_PrintCNT){

        pthread_mutex_lock(&mutex_print);

        printf(temp_PrintSTR, temp_PrintCNT);

        pthread_mutex_unlock(&mutex_print);

}

...

void mutex_SetCNT(int temp_SetCNT){

        pthread_mutex_lock(&mutex_cnt);

        cnt = temp_SetCNT;

        pthread_mutex_unlock(&mutex_cnt);

}

...

int mutex_GetCNT(void){

        int temp_GetCNT;

        pthread_mutex_lock(&mutex_cnt);

        temp_GetCNT = cnt;

        pthread_mutex_unlock(&mutex_cnt);

        return temp_GetCNT;

}
```

Leonardo Fusser (1946995) February 7th, 2022

Week 3 HW

Computerized Systems Optimization

Serge Hould

Intro multi-threading HW:

Q1) For a motor with a PPR of 400 and a gearbox ratio of 67.5-to-1, the number of pulses per degrees is:

$$Number\ of\ pulses\ in\ one\ full\ revolution = 400 * 67.5 = 27'000$$

$$Number\ of\ pulses\ in\ one\ degree = \frac{27'000\ total\ pulses\ in\ one\ full\ revolution}{360^\circ\ for\ one\ full\ revolution} = \mathbf{75\ tics\ per\ degree}$$

Q3) For a robot arm with a range of 140 degrees that generates up to 4000 pulses, the control resolution is:

$$Control\ resolution = \frac{140^\circ\ maximum\ range}{4'000\ maximum\ pulses} = \mathbf{0.035\ degrees\ per\ tic}$$

Q5) For a PV value of 900 tics and a SP value of 1500 tics:

$$Current\ error\ (e) = 1500 - 900 = 600\ tics.\ \frac{600\ tics}{75\ tics\ per\ degree} = \mathbf{8\ degrees}$$

$$Current\ position\ in\ degrees = 900\ tics.\ \frac{900\ tics}{75\ tics\ per\ degree} = \mathbf{12\ degrees}$$

$$Final\ position\ in\ degrees = 1'500\ tics.\ \frac{1'500\ tics}{75\ tics\ per\ degree} = \mathbf{20\ degrees}$$

Q6) The reason why non-volatile should be used for quadrature encoders but not for absolute encoder:

- Non-volatile memory, like EEPROM, should be used for quadrature encoders because they are not able to retain their motor position when power is cycled. On the other hand, non-volatile memory is not needed for absolute encoders because they are able to retain their motor position when power is cycled.

Leonardo Fusser (1946995) February 12th, 2022

Week 4 HW

Computerized Systems Optimization

Serge Hould

Mover4 Robot Arm HW:

Q1) The minimum frequency that the Mover4 servo joints should be updated:

- Every 50mS (or 20Hz), the Mover4 should receive eight new CAN packets for the servo joints.

Q2a) When the Mover4 servo joints are not updated regularly:

- The CAN controller board will prevent any possible damage or harm to the unit and operator by entering into an error state.

Q2b) For each servo joint on the Mover4:

- Inside contents:
  - A motor with a position encoder (the servo itself).
  - A motor controller.

Q3) How the latest Mover4 joint position is restored at power on:

- The Mover4 is able to restore all the joint positions every power cycle thanks to the use of non-volatile memory. Although, this assumes that the joint positions on the Mover4 are not moved when power is lost. If this is the case, regardless of type of memory used, the joint positions on the Mover4 would be lost.

Q4) Explain the following captured CPR-CAN frames:

- **[CPR-CAN packet A]** `0x40 – 0x04 0x80 0x7E 0x40 0x00 0x00`
  - **JointID:** the selected joint is the wrist joint on Mover4 (**0x40**).
  - **Command:** the SetJoint command is used (**0x04**).
  - **Velocity:** velocity of **0x80**.
  - **posH and posL:** the set point (SP) is **0x7E40**.
  - **timeStamp:** there is no timestamp (**0x00**).
  - **digitalout:** the gripper is not enabled and there are no digital outputs set (**0x00**).

- **[CPR-CAN packet B]** `0x41 – 0x00 0x80 0x7C 0x30 0x00 0x1F 0xFF 0x00`
  - **JointID:** feedback coming from the wrist joint on Mover4 (0x40 + 1 = **0x41**).
  - **ErrorCode:** there are no errors being reported (**0x00** – *no bits are set*).
  - **Velocity:** velocity of **0x80**.
  - **posH and posL:** the current position (PV) is **0x7C30**.
  - **shunt:** shunt of **0x00**.
  - **timeStamp:** the time stamp of the command is **0x1F**.
  - **divValue:** divValue of **0xFF**.
  - digitalInputs: there are no digital inputs set (**0x00**).

- **[CPR-CAN packet C]** `0x31 – 0x10 0x80 0x7D 0x00 0x00 0x1F 0xFF 0x00`
  - **JointID:** feedback coming from the elbow joint on Mover4 (0x30 + 1 = **0x31**).
  - **ErrorCode:** there is position lag being reported (**0x10** – *bit4 is set*).
  - Velocity: velocity of 0x80.
  - **posH and posL:** the current position (PV) is **0x7D00** (position zero).
  - **shunt**: shunt of **0x00**.
  - **timeStamp:** the time stamp of the command is **0x1F**.
  - **divValue:** divValue of **0xFF**.
  - **digitalInputs:** there are no digital inputs set (**0x00**).

- **[CPR-CAN packet D]** `0x40 – 0x04 0x80 0x7D 0x21 0x00 0x02`
  - **JointID:** the selected joint is the wrist joint on Mover4 (**0x40**).
  - **Command:** the SetJoint command is used (**0x04**).
  - **Velocity:** velocity of **0x80**.
  - **posH and posL:** the set point (SP) is **0x7D21**.
  - **timeStamp:** there is no timestamp (**0x00**).
  - **digitalout:** the gripper is set to close, and no other digital outputs are set (**0x02**).

Q5a) From the previous captured CPR-CAN packets, find the set point (SP) and current position (PV) in degrees:

- For the previous captured CPR-CAN packet of **A**:
  - The set point (SP) is 0x7E40.

$$Position\ in\ tics = 0x7E40 - 0x7D00 = 0x140\ (320\ tics)$$
$$Position\ in\ degrees = \frac{320\ tics}{65\ tics\ per\ degree} = +4.92\ degrees$$

- For the previous captured CPR-CAN packet of **B**:
  - The current position (PV) is 0x7C30.

$$Position\ in\ tics = 0x7C30 - 0x7D00 = -0xD0\ (-208\ tics)$$
$$Position\ in\ degrees = \frac{-208\ tics}{65\ tics\ per\ degree} = -3.2\ degrees$$

- For the previous captured CPR-CAN packet of **C**:
  - The current position (PV) is 0x7D00.

$$Position\ in\ tics = 0x7D00 - 0x7D00 = 0x0\ (0\ tics)$$
$$Position\ in\ degrees = \frac{0\ tics}{65\ tics\ per\ degree} = 0\ degrees$$

- For the previous captured CPR-CAN packet of **D**:
  - The set point (SP) is 0x7D21.

$$Position\ in\ tics = 0x7D21 - 0x7D00 = 0x21\ (33\ tics)$$
$$Position\ in\ degrees = \frac{33\ tics}{65\ tics\ per\ degree} = +0.50\ degrees$$

Q5b) From the previous captured CPR-CAN packets, find the state of the gripper.

- For the previous captured CPR-CAN packet of **A**:
  - The "`digitalout`" parameter is set to 0x00.
    - Which means that the gripper is not enabled.
- For the previous captured CPR-CAN packet of **B**:
  - For this CPR-CAN packet, the state of the gripper is not evident.
- For the previous captured CPR-CAN packet of **C**:
  - For this CPR-CAN packet, the state of the gripper is not evident.
- For the previous captured CPR-CAN packet of **D**:
  - The "`digitalout`" parameter is set to 0x02.
    - Which means that the gripper is set to close.

Q6) For the following CPR-CAN packet at time 0.0008:

```
0x20 - 0x04 0x80 0x7A 0x67 0x1D 0x00
```

- The set point (SP) value in degrees:
  - Set point (SP) is 0x7A67.

$$Position\ in\ tics = \ 0x7A67 - 0x7D00 = -0x299\ (-665\ tics)$$
$$Position\ in\ degrees = \frac{-665\ tics}{65\ tics\ per\ degree} = -10.23\ degrees$$

- Name of the servo joint:
  - This CPR-CAN packet belongs to the shoulder servo joint on Mover4 (0x20).

Q7) For the following CPR-CAN packet at time 0.0025:

```
0x21 - 0x00 0x80 0x79 0xEF 0x3C 0x1D 0xFF 0x00
```

- The current position (PV) value in degrees:
  - Current position (PV) is 0x79EF.

$$Position\ in\ tics = \ 0x79EF - 0x7D00 = -0x311\ (-785\ tics)$$
$$Position\ in\ degrees = \frac{-785\ tics}{65\ tics\ per\ degree} = -12.07\ degrees$$

- The difference between set point (SP) and current position (PV):
  - Set point (SP) is -10.23 degrees.
  - Current position (PV) is -12.07 degrees.
    - The difference (error) is 1.84 degrees.

Leonardo Fusser (1946995) February 21st, 2022

Week 5 HW

Computerized Systems Optimization

Serge Hould

Sensors for Mechatronics and Robotics (Part 1) HW:

Q1.1)

a) Value for R2 given that R1 is 22kΩ to set the threshold of the comparator to 63.3%:

$$0.633 = \frac{R_2}{R_1 + R_2}$$

$$0.633 = \frac{R_2}{22k\Omega + R_2}$$

$$0.633(22k + R_2) = R_2$$

$$0.633(22k + R_2) * 1'000 = R_2 * 1'000$$

$$633(22k + R_2) = 1'000R_2$$

$$13'926'000 + 633R_2 = 1'000R_2$$

$$13'926'000 - 13'926'000 + 633R_2 = 1'000R_2 - 13'926'000$$

$$633R_2 = 1'000R_2 - 13'926'000$$

$$633R_2 - 1'000R_2 = -13'926'000 + 1'000R_2 - 1'000R_2$$

$$\frac{-367R_2}{-367R_2} = \frac{-13'926'000}{-367R_2}$$

$$R_2 = 37.94k\Omega$$

b) Time to trigger comparator if C = 1nF, 100nF and 1uF given that R3 is 10kΩ:

$$When\ C\ is\ 1nF, \tau = 10k\Omega * 1nF = 10uS$$

$$When\ C\ is\ 100nF, \tau = 10k\Omega * 100nF = 1mS$$

$$When\ C\ is\ 1uF, \tau = 10k\Omega * 1uF = 10mS$$

Q1.2) Complete firmware for capacitive transducer:

```
/* Macros. */
#define DISCHARGE            _LATA0
#define CAPACITANCE_CONST    8.85e-12 //Derived from capacitance formula.
#define CAPACITANCE_AREA     6e-4    //Capacitor plate area (6e-4 = 2cm * 3cm).
#define CAPACITANCE_RESIS    10000   //10kOhm resistor in circuit.
#define METER                1e6    //Converts result into meters.

/* Globals. */
int of = 0, ticks = 0;

void main(void){
     float tau, capacitance, distance; //Variables for calculating distance.
     DISCHARGE = 1;                    //Transistor is ON.
     init_sys();
     enable_interrupt();
     while(1){
          tau = (float)ticks;                    //Reads the ticks.
          capacitance = tau/CAPACITANCE_RESIS;   //Calculates capacitance.
          distance = CAPACITANCE_CONST * CAPACITANCE_AREA / capacitance;
          distance = distance * METER;           //Distance in meters.
     }
}
```

...continues on next page...

```
/* Timer2 is called every 100mS. */
/* Tick is 1 microsecond. */
/* PR2 = 0xffff. */
void __ISR T2InterruptHandler(void){
      IFS0bits.T2IF = 0;    //Clears Timer2 Interrupt flag.
      of++;
}

/* Timer3 is called every second. */
void __ISR T3InterruptHandler(void){
      IFS0bits.T3IF = 0;
      DISCHARGE = 1;        //Transistor is ON (capacitor discharging).
      delay_us(100);        //Small delay for 100uS.
      DISCHARGE = 0;        //Transistor is OFF (capacitor charging).
      TMR2 = 0;             //Resets Timer2.
      of = 0;               //Resets overflow count.
}

/* Change notification interrupt service routine. */
void __ISR CNInterrupt(void){
      /* Positive edge only. */
      if(PORTGbits.RG7){
            ticks = (of*0x10000)+ TMR2;
      }
      IFS1bits.CNIF=0;
}
```

Q2) Complete snippet of code to dump measured distance on a screen every second and to toggle an LED:

$$General\ ADC\ formula: \frac{Max\ ADC\ value}{ADC\ reference} = \frac{Current\ ADC\ value}{Vout}$$

$$Derived\ for\ BBB\ ADC: \frac{4'095}{1.8V} = \frac{Current\ ADC\ value}{Vout}$$

$$Vout = \frac{Current\ ADC\ value}{\left(\frac{4'095}{1.8V}\right)} = \frac{Current\ ADC\ value}{2'275}$$

```
int main(void){
     int rx_ADC;                              //Variable for reading ADC.
     float distance, Vout;                    //Variables for calculation.
     while(1){
          rx_ADC = adc_read();                //Reads the ADC.
          Vout = (float)rx_ADC / 2275;        //Derived from ADC formula.
          distance = 5.5 / Vout;              //Calculates distance in cm.

          printf("Measured distance: %.2fcm", &distance);  //Prints.

          if(distance <= 10){                 //If distance <= 10...
               LED_on();                      //LED stays on.
          }
          else{                               //Else...
               LED_off();                     //LED stays off otherwise.
          }
     }
}
```

...continues on next page...

```
int adc_read(void){
	int fd;			//File pointer.
	char val []=" ";	//Holds up to 4 digits for ADC value.
	fd = open("/sys/ iio /devices/iio:device0/in_voltage2_raw", , O_RDONLY);
	read(fd, &val, 4);	//Read ADC input val (4 digits 0 4095).
	close(fd);		//Close file and stop reading.
	return atoi(val);	//Returns an integer value.
} //End read ADC().

void LED_on(void){
	system("echo 1 > /sys/class/leds/beaglebone:green:usr0/brightness");
}

void LED_off(void){
	system("echo 0 > /sys/class/leds/beaglebone:green:usr0/brightness");
}
```

Q3) Complete snippet of code to measure a distance every 100mS using the HC -SR04 ultrasonic sensor:

```
/* Macros. */
#define TRIG _LATB6   //Triggering pulse pin.

/* Globals. */
int of = 0, ticks = 0;

void main(void){
      float distance;  //Variable for calculating distance.
      init_sys();
      enable_interrupts();
      while(1){
            /* Calculates distance in cm. */
            distance = (float)ticks / 58;
      }
}

/* Timer2 is called every 100mS. */
void __ISR T2InterruptHandler(void){
      IFS0bits.T2IF = 0;
      TRIG = 1;         //Begin of triggering pulse.
      delay_us(10);     //Small delay for 10uS.
      TRIG = 0;         //End of triggering pulse.
}
```

...continues on next page...

```
/* Timer3 is called every 10mS. */
/* Tick is 1 microsecond. */
/* PR3 = 0xffff. */
void __ISR T3InterruptHandler(void){
     IFS0bits.T3IF = 0;
     of++;                              //Increases overflow count.
}

/* Change notification interrupt service routine. */
void __ISR CNInterrupt(void){
     /* Positive edge only. */
     if(PORTGbits.RG7){
          of = 0;                       //Resets overflow count.
          TMR3 = 0;                     //Resets Timer3.
     }
     /* Falling edge only. */
     if(!PORTGbits.RG7){
          ticks = (of*0x10000)+ TMR3;   //Calculates ticks.
     }
     IFS1bits.CNIF = 0;
}
```

Week 6 HW

Computerized Systems Optimization

Serge Hould

Cancelling threads HW:

Q1) Console output:

```
Creating thread2
111111-looping-1111111
2222222222222222 looping 2222222222222222
111111-looping-1111111
2222222222222222 looping 2222222222222222
cancel thread2
111111-looping-1111111
111111-looping-1111111
Creating thread2
111111-looping-1111111
2222222222222222 looping 2222222222222222
111111-looping-1111111
2222222222222222 looping 2222222222222222
cancel thread2
...
```

Week 8 HW

Computerized Systems Optimization

Serge Hould

Robot Arm & Kinematics for Robot Arm HW:

Q1.1) For a 3DOF robotic arm, the base height is **1 inch long**, the humerus length is **4 inches long**, the arm length is **5 inches long** and that the gripper length is **1 inch long**.

Additionally, the gripper angle is set for **0 degrees** and the tip of the gripper's coordinates are given as **x = 4 inches**, **y = 4 inches**, and **z = 6 inches**.

a) Find the shoulder, elbow, and wrist angle values:

Note: *if points in quadrant 1 -> use tan^{-1}(y/x).*
if points in quadrant 2 or 3 -> use tan^{-1}(y/x) + 180° (for degrees) or + π (for radians).
*if points in quadrant 4 -> use tan^{-1}(y/x) + 360° (for degrees) or + 2*π (for radians).*
*if x = 0 and y < 0 -> angle is 270° (for degrees) or 3*π/2 (for radians).*
if x = 0 and y > 0 -> angle is 90° (for degrees) or π/2 (for radians).
if x < 0 and y = 0 -> angle is 180° (for degrees) or π (for radians).
*if x > 0 and y = 0 -> angle is 360° or 0° (for degrees) or 2*π or 0 (for radians).*

$$baseAngle = tan^{-1}\left(\frac{y}{x}\right) = tan^{-1}\left(\frac{4}{4}\right) = 45°$$

$$r = \sqrt{x^2 + y^2} = \sqrt{4^2 + 4^2} = 5.65\ inches$$

$$grip_{off_z} = -\sin(gripAngle) * gripLength = -\sin(0°) * 1 = 0\ inches$$

$$grip_{off_r} = \cos(gripAngle) * gripLength = \cos(0°) * 1 = 1\ inch$$

$$z' = (z - grip_{off_z}) - baseHeight = (6 - 0) - 1 = 5\ inches$$

$$r' = r - grip_{off_r} = 5.65 - 1 = 4.65\ inches$$

$$2h = \sqrt{z'^2 + r'^2} = \sqrt{5^2 + 4.65^2} = 6.82\ inches$$

$$a1 = tan^{-1}\left(\frac{z'}{r'}\right) = tan^{-1}\left(\frac{5}{4.65}\right) = 47.07°$$

$$a2 = cos^{-1}\left(\frac{humLength^2 - armLength^2 + (2h)^2}{2 * humLength * 2h}\right) = cos^{-1}\left(\frac{4^2 - 5^2 + 6.82^2}{2 * 4 * 6.82}\right) = 46.56°$$

$$shldAngle = a1 + a2 = 47.07° + 46.56° = \mathbf{93.63°}$$

$$elbAngle = cos^{-1}\left(\frac{humLength^2 + armLength^2 - (2h)^2}{2 * humLength * armLength}\right) = cos^{-1}\left(\frac{4^2 + 5^2 - 6.82^2}{2 * 4 * 5}\right) = \mathbf{97.92°}$$

$$wristAngle = 360 - gripAngle - elbAngle - shldAngle = 360° - 0° - 97.92° - 93.63° = \mathbf{168.45°}$$

b) Find the corrected shoulder, elbow, and wrist angle values for the Mover4 robot arm:

$$shldAngle(corrected) = 90° - 93.63° = \mathbf{-3.63°}$$

$$elbAngle(corrected) = 180° - 97.92° = \mathbf{82.08°}$$

$$wristAngle(corrected) = 180° - 168.45° = \mathbf{11.55°}$$

Q1.2) The angles of the 3DOF robotic arm from the previous question are **shoulder angle = 92.77°**, **elbow angle' = +62.71°** and **wrist angle' = 0°**. Determine the r-z position for the gripper:

Note: base height = **1 inch**, humerus length = **4 inches**, arm length = **5 inches** and gripper length = **1 inch**.

$$R = humerus\ length * \cos(shoulder\ angle) + arm\ length * \cos(shoulder\ angle - elbow\ angle') + gripper\ length * \cos(shoulder\ angle - elbow\ angle' - wrist\ angle')$$

$$R = 4 * \cos(92.77°) + 5 * \cos(92.77° - 62.71°) + 1 * \cos(92.77° - 62.71° - 0°)$$

$$R = 4.99\ inches$$

$$Z' = humerus\ length * \sin(shoulder\ angle) + arm\ length * \sin(shoulder\ angle - elabow\ angle')$$

$$Z' = 4 * \sin(92.77°) + 5 * \sin(92.77° - 62.71°)$$

$$Z' = 6.49\ inches$$

$$Grip_{OFF_Z} = gripper\ length * \sin(shoulder\ angle - elbow\ angle' - wrist\ angle')$$

$$Grip_{OFF_Z} = 1 * \sin(92.77° - 62.71° - 0°)$$

$$Grip_{OFF_Z} = 0.50\ inches$$

$$Z = Z' + Grip_{OFF_Z} + base\ height$$

$$Z = 6.49 + 0.50 + 1$$

$$Z = 7.99\ inches$$

Q1.3) Function that converts angles in degrees to radians:

```
/* Function to convert degrees to radians. */
double to_radians(double temp_degrees){
     double temp_radians;                             //Temp value.
     temp_radians = temp_degrees * (3.14 / 180.0);    //Converts.
     return temp_radians;                             //Returns.
}
```

Q1.4) Functions that converts angles in radians to degrees:

```
/* Function to convert radians to degrees. */
double to_degrees(double temp_radians){
     double temp_degrees;                             //Temp value.
     temp_degrees = temp_radians * (180.0 / 3.14);    //Converts.
     return temp_degrees;                             //Returns.
}
```

Q1.5) Write a function `to_cart()` that converts angles to x-y-z format (forward kinematics). The angles that are passed to the function are `base_angle`, `shld_angle’`, `elb_angle’` and `wri_angle’`.

Note: the following function utilizes the `to_radians()` function that was created in one of the previous two questions.

```
...
/* Local struct declaration. */
static kin_i GripperPosition;        //Stores gripper X, Y and Z positions.
...
/* Forward kinematics function. */
kin_i to_cart(double base_angle, double shld_angle’, double elb_angle’, double wri_angle’){

        /* Temp variables. */
        double r, z, y, x;
        double grip_off_z, z_prime;

        /* Forward kinematic calculations. */
        r = HUMERUS * cos(to_radians(90 - shld_angle’)) + ULNA * cos(to_radians(90 - shld_angle’
        elb_angle’)) + GRIPPER * cos(to_radians(90 - shld_angle’ – elb_angle’ – wri_angle’));

        z_prime = HUMERUS * sin(to_radians(90 - shld_angle’)) + ULNA * sin(to_radians(90 - shld_angle’
        – elb_angle’));

        grip_off_z = GRIPPER * sin(to_radians(90 - shld_angle’ – elb_angle’ – wri_angle’));
        z = z_prime + grip_off_z + BASE_HGT;
        y = r * sin(to_radians(base_angle));
        x = r * cos(to_radians(base_angle));

        /* Store gripper position in struct. */
        GripperPosition.data[0] = x;  //Stores X position.
        GripperPosition.data[1] = y;  //Stores Y position.
        GripperPosition.data[2] = z;  //Stores Z position.

        /* Returns. */
        return GripperPosition;
}
...
```

Q2.2) C code snippet to move a robot arm back and forth vertically between z = 0 and z = 12 (assuming a grip angle of 45°):

```
...
/* Local struct declaration. */
static kin_f JointAngles;  //Stores gripper X, Y and Z position.
...
/* Moves arm forward vertically (in 0.1 steps). */
for(float z_axis = 0; z_axis <= 12; z_axis += 0.1){
      /* Sets x,y,z angles and grip angle. */
      JointAngles = to_angle(7.0, 9.0, z_axis, 45.0);
      set_all_sp_angles(JointAngles);
      /* Speed of movement (120 steps * 100mS = 12 seconds). */
      delay_ms(100);
}
...
/* Moves arm backward vertically (in 0.1 steps). */
for(float z_axis = 12; z_axis >= 0; z_axis -= 0.1){
      /* Sets x,y,z angles and grip angle. */
      JointAngles = to_angle(7.0, 9.0, z_axis, 45.0);
      set_all_sp_angles(JointAngles);
      /* Speed of movement (120 steps * 100mS = 12 seconds). */
      delay_ms(100);
}
...
```

Q4) For the AL5D robot:

- The AL5D robot has 5 DOFs (degree of freedoms) because it has five joints. Those being the shoulder joint, the elbow joint, the wrist joint, the hand joint and the gripper joint. This does not consider the base joint.

Q5) A manipulator is kinematically redundant:

- When it possesses more degree of freedoms (DOFs) than it minimally needs to execute any given task. This is usually true when the manipulator has 6 DOFs or more.

Q6) A forward kinematics problem:

- A forward kinematics problem occurs when the angles of the joints are given, but the desired gripper position in cartesian space (x, y, and z) is **not** given.

Q7) A reverse kinematics problem:

- A reverse kinematics problem occurs when the angles of the joints are **not** given, but only the desired gripper position in cartesian space (x, y, and z) is given instead.

Q8a) When a robot makes a movement in the x-y plane, which one(s) of the x-y-z variable(s) is (are) constant?

- The z variable is constant when the robot makes a movement in the x-y plane (top view).

Q8b) When a robot makes a circular movement in the y-z plane, which one(s) of the x-y-z variable(s) is (are) constant?

- The x variable is constant when the robot makes a movement in the y-z plane (side view).

Q9) For the Mover4 robot:

a) Give an example of a sensor integrated inside the robot:
   - A sensor that can be found inside the robot is a photoelectric sensor in the encoder circuitry for each of the servo motors inside the joints of the robot. Another sensor that can be found inside the robot is the encoder itself for each of the server motors inside the joints of the robot.

b) What kind of actuator is integrated inside the robot?
   - Multiple servo motors can be found inside the robot for the individual joints of the robot.

c) What kind of robot is it?
   - This is a manipulator robot.

d) What is its DOF?
   - The Mover4 has a 4DOF.

Leonardo Fusser (1946995) March 27th, 2022

Week 10 HW

Computerized Systems Optimization

Serge Hould

Sensors for Mechatronics & Robotics (Part 2) HW:

Q1) For the following thermistor graph:

- The thermistor has a NTC because the line shows that the thermistor has a negative temperature coefficient.

Q2) A TMP61 is connected to an MCU. Assume that V_{BIAS} is 3.3V and R_{BIAS} is 12kΩ, and that the ADC reference voltages are 0V and 3V. For an ambient temperature of -15°C:

a) The value for V_{TEMP} is:

- When the temperature is around -15°C, the TMP61 temperature graph shows that R_{THERM} should be around 7.5kΩ.

  Therefore:

$$V_{TEMP} = \frac{V_{BIAS} * R_{THERM}}{R_{BIAS} + R_{THERM}} = \frac{3.3V * \sim 7.5k}{12k + \sim 7.5k} = \sim 1.26V$$

b) The ADC value for a 10-bit ADC for this temperature is:

$$ADC = \frac{V_{TEMP} * 1'024}{3V} = \frac{1.26V * 1'024}{3V} = \sim 430$$

Q3) A TMP61 is connected to an MCU. Assume that V_{BIAS} is 3.3V and R_{BIAS} is 12kΩ, and that the ADC reference voltages are 0V and 3V. For a MCU that reads an ADC value of 411:

a) The value for V_{TEMP} is:

$$V_{TEMP} = \frac{ADC * 3V}{1'024} = \frac{411 * 3V}{1'024} = 1.20V$$

b) The value for R_{THERM} is:

$$R_{THERM} = \frac{V_{TEMP}}{(V_{BIAS} - V_{TEMP})/_{R_{BIAS}}} = \frac{1.20V}{(3.3V - 1.20V)/_{12k}} = {\sim}6.8k\Omega$$

c) The temperature is (from the partial software LUT of the TMP61):

- The temperature is around -31°C.

Q5) Complete snippet of code for a digital thermometer using the TMP61 thermistor. Assume that V_{BIAS} is 3.3V and R_{BIAS} is 10kΩ. Also, the ADC has a 10-bit range, and the reference voltages are 0V and 3V.

a) Using the Steinhart-Hart equation (assume A = 60.38e-3, B = -83.58e-4 and C = 255.18e-7):

```
/* Includes. */
#include <math.h>

/* Macros. */
#define RBIAS    10000
#define VBIAS    3.3
#define A        60.38e-3
#define B        -83.58e-4
#define C        255.18e-7
#define KELVIN   273.15

/* Globals. */
int rx_ADC;
float Vtemp, Rtherm, Celsius;

void main(void){
      adc_init();
      while(1);
}
```

...continues on next page...

```
/* Called every second. */
void __ISR( _TIMER_2_VECTOR, IPL1SOFT) T2InterruptHandler(void){
      IFS0bits.T2IF = 0;    //Clears interrupt flag.

      /* Reads from ADC and calculates Rtherm. */
      rx_ADC = adc_read();
      Vtemp = (float)rx_ADC * 3 / 1024;
      Rtherm = (Vtemp / (VBIAS – Vtemp) / RBIAS);

      /* Calculates temperature in celsius. */
      Celsius = A + B * (log(Rtherm)) + C * pow(log(Rtherm), 3);
      Celsius = 1 / Celsius;
      Celsius = Celsius – KELVIN;
}
```

b) Using a lookup table with a limited range:

```
/* Macros. */
#define RBIAS    10000
#define VBIAS    3.3

/* Globals. */
int rx_ADC;
float Vtemp, Rtherm, Celsius;

void main(void){
      adc_init();
      while(1);
}

/* Called every second. */
void __ISR( _TIMER_2_VECTOR, IPL1SOFT) T2InterruptHandler(void){
      IFS0bits.T2IF = 0;    //Clears interrupt flag.

      /* Reads from ADC and calculates Rtherm. */
      rx_ADC = adc_read();
      Vtemp = (float)rx_ADC * 3 / 1024;
      Rtherm = (Vtemp / (VBIAS – Vtemp) / RBIAS);

      /* Calculates temperature in Celsius. */
      for(int i = 0; Rtherm <= table[i][1]; i++){
            Celsius = table[i][1];
     }
}
```

Q6) Find the ambient temperature if the output of a MCP9700 is 100mV:

$$T_a = \frac{(100mV - 500mV)}{10mV} = -40°C$$

Q7) Complete snippet of code for a digital thermometer using the MCP9700 active thermistor:

```
/* Globals. */
int rx_ADC;
float Vout, Celsius;

void main(void){
     adc_init();
     while(1);
}

/* Called every second. */
void __ISR( _TIMER_2_VECTOR, IPL1SOFT) T2InterruptHandler(void){
     IFS0bits.T2IF = 0;    //Clears interrupt flag.

     /* Reads from ADC and calculates Vout. */
     rx_ADC = adc_read();
     Vout = (float)rx_ADC * 3 / 1024;

     /* Calculates temperature in Celsius. */
     Celsius = (Vout - 500e-3) / 10e-3;
}
```

Q8) Match the proper sensor with the proper application:

- Ultrasonic range finder: *detect objects up to a few meters*.
- Photoelectric proximity sensors: *detect objects of less than one meter*.
- Capacitive proximity sensors: *measure displacements in the micrometer range*.

Q9) To implement a temperature sensor, of the following given sensors, which one is the easiest to implement:

1. An active thermistor integrated circuit.
2. A PTC thermistor.
3. A NTC thermistor.

- The easiest one to implement is the active thermistor integrated circuit. This is because it reads the temperature in a linear range. The other two types of sensors either require a lookup table or some form of lengthy math equations to derive the temperature.