```
1  ------------------------------------------------------------------------
2  This PDF contains the code created by Aurora Moholth, Sarah Jane Sandell & Thale Gartland, for the project in
   real-time systems.
3  ------------------------------------------------------------------------
4
5
6  ------------------------------------------------------------------------
7  control_program.ads
8  ------------------------------------------------------------------------
9  -- This program is intended to determine how the meacanum car reacts when there is a obejct in front of the vechicle
   or when it overturn.
10
11 package control_program is
12
13    task Control_Car with Priority => 1;
14
15 end control_program;
16
17
18 ------------------------------------------------------------------------
19 control_program.adb
20 ------------------------------------------------------------------------
21 with AccelerometerTask_pk;
22 with distance_sensor;
23 with Wheels;
24 with Acc_Storage_pk;
25 with distance_sensor_storage_pk;
26 with Ada.Real_Time; use Ada.Real_Time;
27
28 package body Control_Program is
29
30    -- This is the states the car can have: forward, turn_right, turned
31    type move_state is (forward, turn_right, turned);
32
33    -- The task Control_Car is a task that get in the infromation from the sensor task and control the movements to
      the car.
34    -- By processing this data, this task set the state of what the car shall do.
35    -- The task use a case statement to switch between the state to the car.
36
37    task body Control_Car is
38       Car : Wheels.Set_of_wheels;             -- The car variable define the car in wheels.
39       current_state : move_state := forward;  -- Before the loop in the task start the current_state to the car must
         be set to forward.
40
41       -- The variable Time_Now and Time_next is used to control how long the car turns to the right after detection
42       Time_Now : Time;
43       Time_next : Time;                       --
44       D : Time_Span := Milliseconds (1700);   -- The variable D is used to control how long time the car shall turn
         right.
45    begin
46       loop
47          -- This case statement is used to set the states that control the movements to the car.
```

```
48
49          case current_state is
50
51             -- The forward case set the car to drive forward.
52             -- If the accelerometer detect that the car has overturned the current_state is set to turned.
53             -- If the distance sensor detect something in front the Time_Next variable is set to the clock time plus
               the D variable.
54             -- Then the current_state is switched to turn_right.
55             -- Now it has been determined that everything is OK. We repeatedly will tell the wheels to move forward.
56             when forward =>
57                Wheels.Drive_forward(Car);
58                if not(Acc_Storage_pk.storage.get_upright) then
59                   current_state := turned;
60                elsif distance_sensor_storage_pk.Sensor_flag.Get  then
61                   Time_Next := Clock + D;
62                   current_state := turn_right;
63                else
64                   Wheels.Drive_forward(Car);
65                end if;
66
67             -- The turn_right case set the car to rotate clockwise.
68             -- If the accelerometer detect that the car has overturned the current_stat is set to turned.
69             -- If the car dosent overturn the car will rotate until the time_Now is more than Time_Next.
70             -- When Time_Now is more than Time_Next the current_state will switch to forward.
71             when turn_right =>
72                Wheels.Rotate_clockwise(car);
73                if not(Acc_Storage_pk.storage.get_upright) then
74                   current_state := turned;
75                end if;
76                Time_Now := Clock;
77                   if (Time_Now > Time_Next) then
78                   current_state := forward;
79                end if;
80
81             -- The turned case set the car on brake wich mean that the wheels stop rotating.
82             -- If the accelerometer detect that the car is upright then the current_state is set to forward.
83             when turned =>
84                Wheels.Brake(Car);
85                if (Acc_Storage_pk.storage.get_upright) then
86                   current_state := forward;
87                end if;
88          end case;
89
90          delay until Clock + Microseconds(500);
91
92       end loop;
93
94    end Control_Car;
95 end Control_Program;
96
97
98 ---------------------------------------------------------------------
```

```
99  accelerometertask_pk.ads
100 ------------------------------------------------------------------------
101
102 package AccelerometerTask_pk is
103
104    task AccelerometerTask with Priority => 1;
105
106
107 end AccelerometerTask_pk;
108
109
110 ------------------------------------------------------------------------
111 accelerometertask_pk.adb
112 ------------------------------------------------------------------------
113 with LSM303AGR; use LSM303AGR;
114 with MicroBit.Accelerometer;
115 with Ada.Text_IO; use Ada.Text_IO;
116 with Ada.Real_Time; use Ada.Real_Time;
117 use MicroBit;
118 with Acc_Storage_pk;
119
120
121 package body AccelerometerTask_pk is
122
123    -- This task retrieves and processes the data from the accelerometer.
124    -- This task is based on the example for accelerometer in
       ADA_Drivers_Library/examples/MicroBit_v2/ravenscar/accelerometer
125    task body AccelerometerTask is
126       Data: All_Axes_Data;            -- The variable Data collect data from the accelerometer and the type
          All_Axes_Data
127       Threshold : constant := 179;    -- The threshold for when the car is overturned or not is set to 179 for all
          axis.
128       Overturned : Boolean := false;  -- Registrer if the car has overturned
129       Time_Now : Time;
130
131    begin
132
133       loop
134          Time_Now := Clock;
135          Data := MicroBit.Accelerometer.AccelData; -- Read the accelerometer data
136
137          -- The if statement detect the slope of the microbit and then set the overturned variable if the Thershold
             is over 179 or lower than -179
138          -- If the car slope between 179 and -179 in x and y direction, the overturned variable is set false.
139
140          if Data.X > Threshold then      -- If the car slope more than 179 in x direction, then set the overturned
             variable to true.
141             Overturned := True;
142          elsif Data.X < -Threshold then  -- If the car slope more less than 179 in x direction, then set the
             overturned variable to true.
143             Overturned := True;
144          elsif Data.Y > Threshold then   -- If the car slope more than 179 in y direction, then set the overturned
```

```
            variable to true.
145            Overturned := True;
146         elsif Data.Y < -Threshold then  -- If the car slope less than -179 in y direction, then set the overturned
            variable to true.
147            Overturned := False;
148            else
149              Overturned := True;
150         end if;
151
152         -- This line set the the protectet object in the Acc_Storage_pk.
153         -- If the overturned variable is true shall the upright procedure be set to false and virce versa.
154         Acc_Storage_pk.storage.upright(not(Overturned));
155
156         delay until Time_Now + Microseconds(2500);
157
158      end loop;
159
160   end AccelerometerTask;
161 end AccelerometerTask_pk;
162
163
164 ------------------------------------------------------------------------
165 acc_storage_pk.ads
166 ------------------------------------------------------------------------
167 --Acc_Storage_pk is a package that include a protected object.
168
169 package Acc_Storage_pk is
170
171    -- This protected object stores the state to the accelerometer.
172    protected type Acc_Storage_t is
173
174       -- The upright procedure sets the state of the car.
175       -- If the accelerometer is overturned, then the upright function is set false and the car_State is set false.
176        procedure upright( state : in boolean );
177
178       -- The get_upright function returns the state of the car.
179       -- If the function returns a true variable it means that the car is uprigth.
180       -- If the funciton returns a false variable it means that the car has overturned.
181       function get_upright return boolean;
182
183    private
184       car_state : boolean := true;
185    end Acc_Storage_t;
186
187    storage : Acc_Storage_t;
188 end Acc_Storage_pk;
189
190
191 ------------------------------------------------------------------------
192 acc_storage_pk.adb
193 ------------------------------------------------------------------------
194 --This package body includes the decleration to the protected object Acc_Storage_t.
```

```
--The protected object Acc_Storage_t is between the task accelerometerTask and control_program task.

package body Acc_Storage_pk isS
   protected body Acc_Storage_t is

      -- The procedure upright set the car_state varibale to state.
      procedure upright( state : in boolean ) is
      begin
         -- The private variable car_state gets the state value from the procedure.
         car_state := state;
      end upright;

      -- The function get_upright returns the boolean value to car_state.
      function get_upright return boolean is
      begin
         return car_state;
      end get_upright;

   end Acc_Storage_t;
end Acc_Storage_pk;


-------------------------------------------------------------------
distance_sensor.ads
-------------------------------------------------------------------
-- This package provides the mechanisms needed to operate the ultrasonic sensor.

with MicroBit.IOsForTasking;
with Ada.Real_Time; use Ada.Real_Time;
with distance_sensor_storage_pk;

package Distance_sensor is
   -- Sets trigger pin to 1 for 10 signal microseconds to emit ultrasound signal.
   procedure Trigger (Trigger_pin_val : MicroBit.IOsForTasking.Pin_Id);

   -- Sets echo pin to 1 then monitors time taken for rebounded ultrasound signal to cause it to return to 0.
   -- The time taken for the signal to return is used to calculate the distance the signal travelled.
   -- The function returns calculated distance as a floating point value.
   function Echo (Echo_pin_val : MicroBit.IOsForTasking.Pin_Id) return Float;

   -- Declares a task to be used to loop the sensor functions, so the sensor functions can continuously test whether
   the car is too near another object
   task Sensor_loop with Priority => 1;
end Distance_sensor;


-------------------------------------------------------------------
distance_sensor.adb
-------------------------------------------------------------------
--This package bpdy includes the decleration of the procedure Trigger, the function Echo and the task Sensor_loop.
--The protected object is between the task Sensor_loop and the protected object Sensor_flag.

```

```
246  package body Distance_sensor is
247
248     -- This procedure controls the trigger pin on the HC-SR04.
249     -- The input value to the procedure is the pin number.
250     procedure Trigger (Trigger_pin_val : MicroBit.IOsForTasking.Pin_Id) is
251        Signal_duration : constant Ada.Real_Time.Time_Span := Ada.Real_Time.Microseconds(10);
252        Delay_time : Ada.Real_Time.Time := Ada.Real_Time.Clock + Signal_duration;
253     begin
254        MicroBit.IOsForTasking.Set(Trigger_pin_val, True);
255        delay until Delay_time;
256        MicroBit.IOsForTasking.Set(Trigger_pin_val, False);
257     end Trigger;
258
259
260     function Echo (Echo_pin_val : MicroBit.IOsForTasking.Pin_Id) return Float is
261        Initial_time : Ada.Real_Time.Time := Ada.Real_Time.Clock;
262        Final_time : Ada.Real_Time.Time := Ada.Real_Time.Clock;
263        Distance_detected : Float;
264     begin
265        MicroBit.IOsForTasking.Set(Echo_pin_val, True);
266
267        -- Updates Initial_time value until confirmed that echo pin set to TRUE.
268        while MicroBit.IOsForTasking.Set(Echo_pin_val) = False loop
269           Initial_time := Ada.Real_Time.Clock;
270        end loop;
271
272        -- Updates Final_time value until signal confirmed returned, which is indicated by echo pin returning to FALSE.
273        while MicroBit.IOsForTasking.Set(Echo_pin_val) loop
274           Final_time := Ada.Real_Time.Clock;
275        end loop;
276
277        -- Calculates distance travelled using time measured.
278        Distance_detected := Float(To_Duration((34300*(Final_time - Initial_time))/2));
279        return Distance_detected;
280
281     end Echo;
282
283
284     -- This task retrieves and processes the data from the distance sensor HC-SR04.
285     --  It sets the appropriate flag value based on the sensor's input.
286     task body Sensor_loop
       is
287        Time_Now : Time;
288
289     begin
290        loop
291           Time_Now := Clock;  -- Here the Time_Now variable is set to the time of the clock of the start of the
              loop.

292           Trigger(10);        -- Here the Trigger procedure are used and the pin 10 is the input
              variable.
293
```

```
294            -- The if loop use the Echo function to chech if the distance between the sensor and a object in front is
               less than 12cm.
295            if Echo(4) < 12.0 then    -- If the distance is less than 12cm the value of the protected objec sensor_flag
               is set true.
296               distance_sensor_storage_pk.sensor_flag.Set(True);
297            else                      -- If the distance is not less than 12c the value of the protected object
               sensor_flag is set flase.
298
                  distance_sensor_storage_pk.Sensor_flag.Set(False);

299            end if;
300
301            delay until Time_Now + Milliseconds(20);
302         end loop;
303
304      end Sensor_loop;
305   end Distance_sensor;
306
307
308   ------------------------------------------------------------------------
309   distance_sensor_storage_pk.ads
310   ------------------------------------------------------------------------
311   -- This package contains the implementation of the ultrasonic sensor.
312   -- It defines the functions to set the trigger pin and read the echo
313   -- pin and then loops them in a task.
314
315   package distance_sensor_storage_pk is
316         --Declares protected object which indicate if the car is too close to an object.
317      protected Sensor_flag is
318         procedure Set (Value : Boolean);
319         function Get return Boolean;
320
321      private
322         Flag_value : Boolean := False;
323      end Sensor_flag;
324   end distance_sensor_storage_pk;
325
326
327   ------------------------------------------------------------------------
328   distance_sensor_storage_pk.adb
329   ------------------------------------------------------------------------
330   package body distance_sensor_storage_pk is
331
332      protected body Sensor_flag is
333         --To set the value of Flag_value.
334         procedure Set (Value : Boolean) is
335         begin
336            Flag_value := Value;
337         end Set;
338
339         --To return the value of Flag_value.
340         function Get return Boolean is
```

```
      begin
         return Flag_value;
      end Get;

   end Sensor_flag;
end distance_sensor_storage_pk;


-------------------------------------------------------------------
wheels.ads
-------------------------------------------------------------------
-- This package provides the various driving modes for a car. It does so by
-- creating a set of wheels and then setting the individual wheels to either
-- drive forwards, backwards or stop as required.

with Wheel;

package Wheels is

   -- Creates a set of wheels as a type and sets each set
   -- Each number on the pin is in this order: Pwm_pin, Forward_pin, Backward_pin
   type Set_of_wheels is record
      Front_left_wheel : Wheel.Single_wheel  := (0,6,7);
      Front_right_wheel : Wheel.Single_wheel  := (1,12,13);
      Back_left_wheel : Wheel.Single_wheel  := (0,14,15);
      Back_right_wheel : Wheel.Single_wheel  := (1,2,3);
   end record;

   procedure Drive_forward (Self : Set_of_wheels);
   procedure Brake (Self : Set_of_wheels);
   procedure Rotate_clockwise (Self : Set_of_wheels);
end Wheels;


-------------------------------------------------------------------
wheels.adb
-------------------------------------------------------------------
package body Wheels is

   -- Procedure that sets the car's set of wheels to drive forwards
   procedure Drive_forward (Self : Set_of_wheels) is
   begin
      Self.Front_left_wheel.Rotate_forwards;
      Self.Front_right_wheel.Rotate_forwards;
      Self.Back_left_wheel.Rotate_forwards;
      Self.Back_right_wheel.Rotate_forwards;
   end Drive_forward;

   -- Rotates car in clockwise direction on the spot
   procedure Rotate_clockwise (Self : Set_of_wheels) is
   begin
      Self.Front_left_wheel.Rotate_forwards;
```

```
393         Self.Front_right_wheel.Rotate_backwards;
394         Self.Back_left_wheel.Rotate_forwards;
395         Self.Back_right_wheel.Rotate_backwards;
396      end Rotate_clockwise;
397
398      -- Stops all four wheels, causing the car to break
399      procedure Brake (Self : Set_of_wheels) is
400      begin
401         Self.Front_left_wheel.Stop;
402         Self.Front_right_wheel.Stop;
403         Self.Back_left_wheel.Stop;
404         Self.Back_right_wheel.Stop;
405      end Brake;
406   end Wheels;
407
408
409   ------------------------------------------------------------------
410   wheel.ads
411   ------------------------------------------------------------------
412   -- This package contains the functionality for a single wheel. It provides the
413   -- ability to instruct each individual wheel to rotate forwards or backwards and stop.
414
415   with MicroBit.IOsForTasking;
416
417   package Wheel is
418
419      type Single_wheel is tagged record
420         Pwm_pin, Forward_pin, Backward_pin : MicroBit.IOsForTasking.Pin_Id;
421      end record;
422
423      procedure Rotate_forwards (Self: Single_wheel);
424      procedure Rotate_backwards (Self : Single_wheel);
425      procedure Stop (Self : Single_wheel);
426
427   end Wheel;
428
429
430   ------------------------------------------------------------------
431   wheel.adb
432   ------------------------------------------------------------------
433   package body Wheel is
434
435      -- Procedure sets the appropriate pin values for a given wheel such that it rotates forwards
436      procedure Rotate_forwards (Self: Single_wheel) is
437      begin
438         MicroBit.IOsForTasking.Set(Self.Pwm_pin, True);
439         MicroBit.IOsForTasking.Set(Self.Forward_pin, True);
440         MicroBit.IOsForTasking.Set(Self.Backward_pin, False);
441      end Rotate_forwards;
442
443      -- Procedure sets the appropriate pin values for a given wheel such that it rotates backwards
444      procedure Rotate_backwards (Self : Single_wheel) is
```

```
445      begin
446         MicroBit.IOsForTasking.Set(Self.Pwm_pin, True);
447         MicroBit.IOsForTasking.Set(Self.Forward_pin, False);
448         MicroBit.IOsForTasking.Set(Self.Backward_pin, True);
449      end Rotate_backwards;
450
451      -- Sets PWM pin to FALSE so that the given wheel stops
452      procedure Stop (Self : Single_wheel) is
453      begin
454         MicroBit.IOsForTasking.Set(Self.Pwm_pin, False);
455      end Stop;
456   end Wheel;
457
```