Luggage sorter robot documentation

A project by

Boga Karla Maria

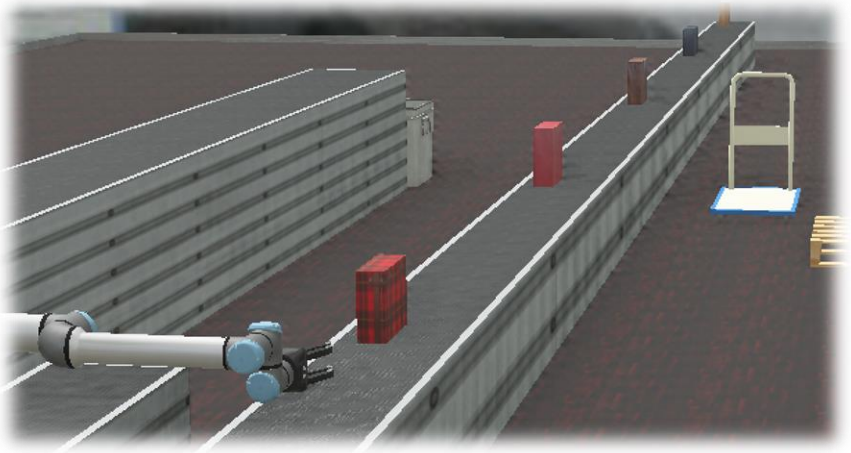
Goța Timeea

Summary

We came up with this idea after watching a video with Heathrow's baggage robot. The video presented how a robot organizes luggage, so we tried to implement a similar idea.

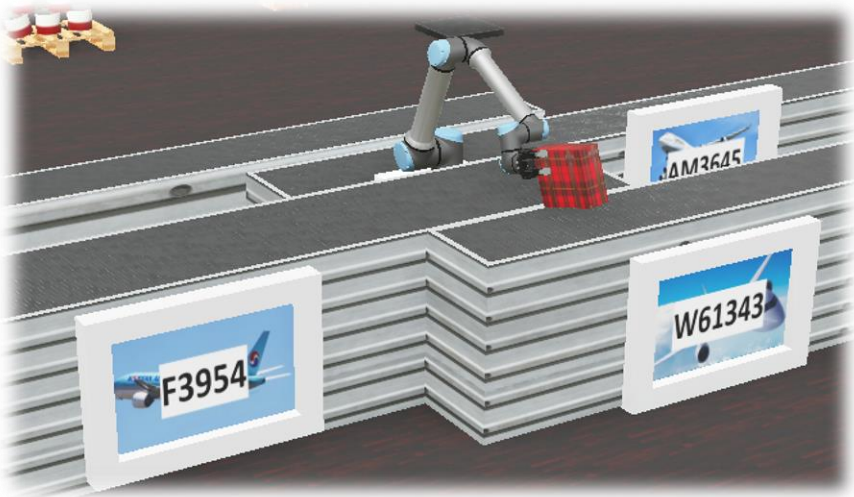
https://www.youtube.com/watch?v=h0_4Xp_XkQ4&ab_channel=HeathrowAirport

Our robot sort luggage for different flights depending on a color tag, each color representing a different flight number. To do this we put a camera with object and color recognition and a distance sensor on the robot.



The baggage come on a conveyor belt. At that time the robot is in a waiting state.

When the sensor detects a short distance between the sensor and the luggage, the robot grasps it and put it on another conveyor belt that transports the object to the box corresponding to the right flight.

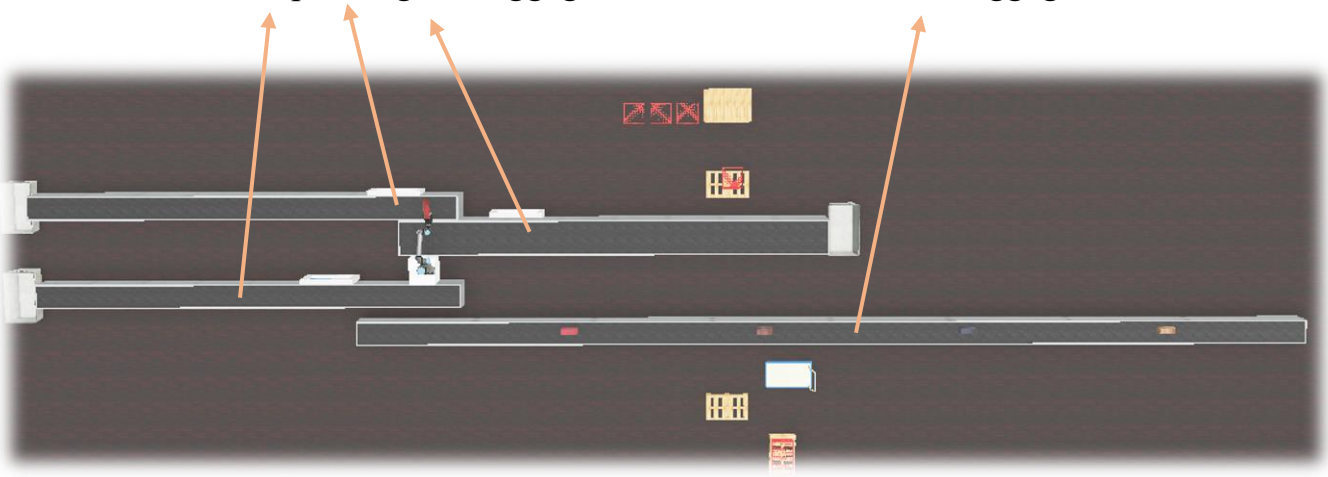



```
start----------------------
Number of luggages: 1.
FLIGHT NUMBER: W61343
Rotating arm back
----------------------end
```

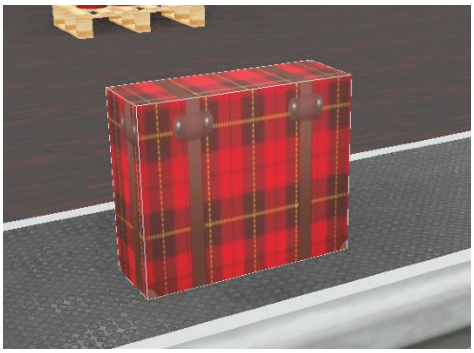
The simulation environment

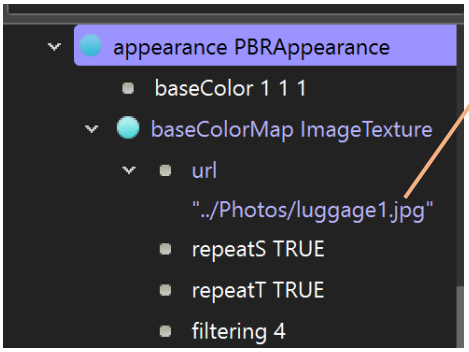
On the rectangle arena we have 4 conveyor belts:

- three for separating the luggage
- one for all luggage

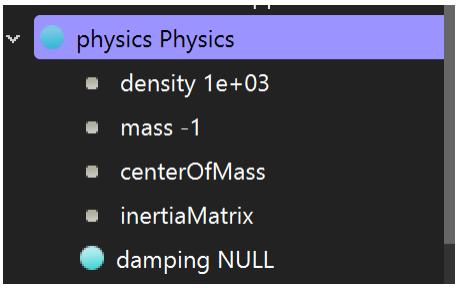


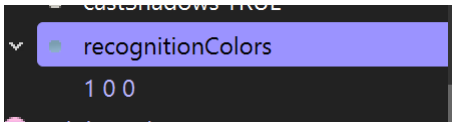
For the luggage we used solid boxes and added the Physics and PBRAppearance nodes.





Here we added the aspect of the baggage.

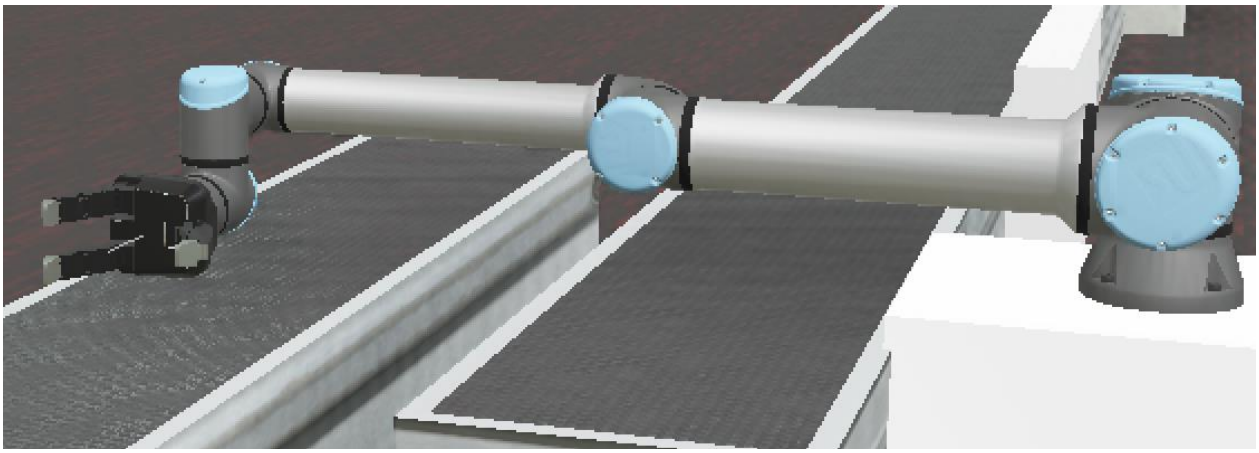




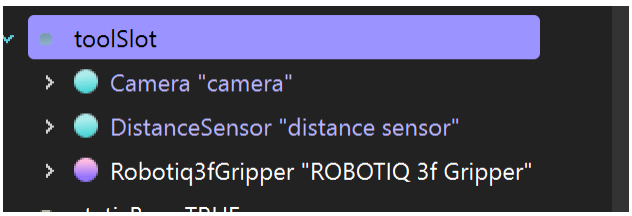
This property allows the camera to detect different colors which represent different flights. In this example the color red.

The robot simulation

We used a UR10e robot (we named him Jimmy) with a Robotiq3fGripper.



As we mentioned earlier, we attached a camera with a recognition node and a distance sensor node.



Camera's point of view

The sensor's ray

The Code

We used C language to program our robot.

In the beginning we include the libraries that we need:

my_controller.c

```c
#include <webots/distance_sensor.h>
#include <webots/motor.h>
#include <webots/position_sensor.h>
#include <webots/robot.h>
#include <webots/camera.h>
#include <webots/camera_recognition_object.h>

#include <stdio.h>

```

In line 12 we declare an enumeration called *State* which represents the 5 stages of movements.

```c
enum State { WAITING, GRASPING, ROTATING, RELEASING, ROTATING_BACK };
```

Here we initialized the robot(line 15), the camera(line 17) and enabled the camera(line 18) and camera recognition(line 19).

```c
int main() {
  wb_robot_init();

  WbDeviceTag camera = wb_robot_get_device("camera");
  wb_camera_enable(camera, 2 * TIME_STEP);
  wb_camera_recognition_enable(camera, 2 * TIME_STEP);

```

In line 22 we defined and initialized the variable *state* with *WAITING*.

From line 22 to 25 we declared three constants named *target_position[], target_position1[], target_position2[]* representing the positions of the joints.

```
21  int i = 0;
22  int state = WAITING;
23  const double target_positions[] = {-1.3, -3, -2.38, -2};
24  const double target_positions1[] = {-2.3, 2.8, -2.31, -1.51};
25  const double target_positions2[] = {-2, -1.6, -2.5, -1.51};
26  double r = 0.00, g = 0.00, b = 0.00;
27  double speed = 1;
```

In line 26 were defined and initialized the three colors used for sorting the luggage. Also, in the next line the variable *speed* is defined and initialized with 1.

The *hand_motors[3]* and *ur_motors[4]* are 2 arrays. The first one is for the gripper and the second one is for the main body of the robot. The *wb_robot_get_device* function is used to retrieve a device with a specified name, and the returned device is then assigned to each element of the *hand_motors[]* array respectively the *ur_motors[]* array.

```
29  WbDeviceTag hand_motors[3];
30  hand_motors[0] = wb_robot_get_device("finger_1_joint_1");
31  hand_motors[1] = wb_robot_get_device("finger_2_joint_1");
32  hand_motors[2] = wb_robot_get_device("finger_middle_joint_1");
33
34  WbDeviceTag ur_motors[4];
35  ur_motors[0] = wb_robot_get_device("shoulder_lift_joint");
36  ur_motors[1] = wb_robot_get_device("elbow_joint");
37  ur_motors[2] = wb_robot_get_device("wrist_1_joint");
38  ur_motors[3] = wb_robot_get_device("wrist_2_joint");
```

Then the *distance_sensor* and the *position_sensor* are initialized and enabled in the following lines.

```
40  WbDeviceTag distance_sensor = wb_robot_get_device("distance sensor");
41  wb_distance_sensor_enable(distance_sensor, TIME_STEP);
42
43  WbDeviceTag position_sensor = wb_robot_get_device("wrist_1_joint_sensor");
44  wb_position_sensor_enable(position_sensor, TIME_STEP);
45
```


Using the *wb_motor_set_velocity* we assign the speed to the motors of the main body.

```
46   for (i = 0; i < 4; ++i)
47     wb_motor_set_velocity(ur_motors[i], speed);
```

To put the robot in an initial state we used 2 “for” loops for the arm and the gripper.

```
50   for (int i = 0; i < 4; ++i) {
51     wb_motor_set_position(ur_motors[i], 0.0);
52   }
53   for (i = 0; i < 3; ++i)
54       wb_motor_set_position(hand_motors[i], 0.05);
```

Now we enter the main loop.

In line 60 we initialize an integer named *number_of_objects* takes, using *wb_camera_recognition_get_objects* function, the value representing the number of objects recognized by the camera.

The *wb_camera_recognition_get_objects(camera)* function declares a pointer “objects” of type *WbCameraRecognitionObject* and initializes it with the result of calling the *wb_camera_recognition_get_objects* function with the camera parameter.

```
56   while (wb_robot_step(TIME_STEP) != -1) {
57
58       printf("start---------------------\n");
59
60       int number_of_objects = wb_camera_recognition_get_number_of_objects(camera);
61       printf("\nNumber of luggages: %d.\n", number_of_objects);
62
63       const WbCameraRecognitionObject *objects = wb_camera_recognition_get_objects(camera);
64
```

```
65      for (i = 0; i < number_of_objects; ++i) {
66        for (int j = 0; j < objects[i].number_of_colors; ++j) {
67          r = objects[i].colors[3 * j];
68          g = objects[i].colors[3 * j + 1];
69          b = objects[i].colors[3 * j + 2];
70          if (r == 1.000000 && g == 0.000000 && b == 0.000000)
71            printf("FLIGHT NUMBER: W61343\n");
72          else if (r == 0.000000 && g == 1.000000 && b == 0.000000)
73            printf("FLIGHT NUMBER: AM3645\n");
74           else if (r == 0.000000 && g == 0.000000 && b == 1.000000)
75            printf("FLIGHT NUMBER: F3954\n");
76        }
77      }
```

This is a nested loop structurure that iterates over the colors of each recognized object. *objects[i].number_of_colors* represents the number of colors associated with the i-th object.

Color Extraction (“*r = objects[i].colors[3 * j];*, *g = objects[i].colors[3 * j + 1*];, b = objects[i].colors[3 * j + 2];”):

- Within the inner loop, these lines extract the red (r), green (g), and blue (b) components of the j-th color associated with the i-th object from the objects array.
- The colors array stores the RGB values of each color associated with the object.

The code then checks the RGB values of the extracted color using a series of if-else if statements. Each color representing a flight number, we print the flight number for each luggage.

Now, we have a *switch* that controls the behavior of the robot.

```
79      switch (state) {
80
81        case WAITING:
82        for (i = 0; i < 3; ++i)
83              wb_motor_set_position(hand_motors[i], 0.05);
84          printf("Distance: %lf\n", wb_distance_sensor_get_value(distance_sensor));
85          fflush(stdout);
86          if (wb_distance_sensor_get_value(distance_sensor) < 300 && wb_distance_sensor_get_value(distance_sensor) > 100) {
87            state = GRASPING;
88            printf("Grasping object\n");
89            for (i = 0; i < 3; ++i)
90              wb_motor_set_position(hand_motors[i], 0.85);
91          }
92          break;
```

The first case is *WAINTING*. The first for sets the robot in the initial position, setting all his motors with *wb_motor_set_position* function. Then we print the distance between the sensor and the object. If the distance is in between the values 100 and 300, the *state* is changing to *GRASPING*. It then enters another loop that sets the position of each motor in *hand_motors* to 0.85 so that the gripper is closing to grab the suitcase.

```
94          case GRASPING:
95            if ( r == 1.000000 && g == 0.000000 && b == 0.000000)
96              for(int i=0; i<4; ++i)
97                  wb_motor_set_position(ur_motors[i], target_positions2[i]);
98              else if (r == 0.000000 && g == 1.000000 && b == 0.000000)
99                    for(i=0; i<4; ++i)
100                     wb_motor_set_position(ur_motors[i], target_positions1[i]);
101
102             else if(r == 0.000000 && g == 0.000000 && b == 1.000000)
103                 for(i=0; i<4; i++)
104                    wb_motor_set_position(ur_motors[i], target_positions[i]);
105           printf("Rotating arm\n");
106           state = ROTATING;
107           break;
```

In the *GRASPING* case we have a color condition that determines where the suitcase should be placed using the wb_motor_set_position function that has two parameters: *ur_motors[]*(the array of the arm's motors) and *target_position[]/ target_position1[]/ target_position2[]* (the constants of the positions where the robot needs to be). Then the *state* changes to *ROTATING*.

```
109         case ROTATING:
110           if (wb_position_sensor_get_value(position_sensor) < -2.3) {
111             printf("Releasing object\n");
112             state = RELEASING;
113             for (i = 0; i < 3; ++i)
114               wb_motor_set_position(hand_motors[i], wb_motor_get_min_position(hand_motors[i]));
115           }
116           break;
```

In the *ROTATING* state we have a condition: if the position sensor is less than -2.3, then the state is changing to *RELEASING* and the gripper's motors come to the initial position using *wb_motor_set_position* function and *wb_motor_get_min_position* function(line 113-114).

```
118         case RELEASING:
119           for (i = 0; i < 4; ++i)
120             wb_motor_set_position(ur_motors[i], 0.0);
121           printf("Rotating arm back\n");
122           state = ROTATING_BACK;
123           break;
```

In the *RELEASING* state the robot's motors are going back to initial position.

```
125         case ROTATING_BACK:
126           if (wb_position_sensor_get_value(position_sensor) > -0.1) {
127             state = WAITING;
128             printf("Waiting object\n");
129           }
130           break;
131       }
132
133     printf("----------------------end\n");
134   }
```

In the *ROTATING_BACK* state we have a condition: if the value of the sensor position is bigger than -0.1, the state is changing back to *WAITING*. This is the end of the *switch* and the *while loop.*

```
136   wb_robot_cleanup();
137   return 0;
138 }
```

wb_robot_cleanup() is essential for ensuring that all resources associated with the robot's operation within the simulation environment are properly released and cleaned up before the program exits.

Extending this project could involve:

- Using multiple robots to build bigger projects.
- Integrating more sensors.
- Integrating other sorting criteria, such as mass or shape.

Bibliography:

https://cyberbotics.com/doc/guide/tutorials

https://cyberbotics.com/doc/guide/index

https://www.youtube.com/watch?v=1CN6OZIgwT0&ab_channel=KajalGada

https://www.youtube.com/watch?v=BuQazGSChaw&t=443s&ab_channel=Softillusion

https://www.youtube.com/watch?v=0Ts30Zi4U1Q&t=3s&ab_channel=VenkataVaibhavBhagavatula