Robotics (U30241)

Coloured object sorting using a mobile robot

Samuel Renshaw-Panting (UP863457)

1 Introduction

The aim of this project is to design and improve the performance of object manipulation using a mobile robot and then apply this development to solve a real world problem in a simulated scenario using the v-rep software.

The problem that will be the focal point of this project is sorting objects in industrial environments using robotics to automate the process.

There are a number of issues that can arise when using robotics to sort objects in industrial environments, mainly due to their inherently dynamic nature. Places like factories, e-commerce centers or shipping centers for postal vans are examples of such environments.

There are two clear issues relating to the problem of sorting objects in industrial environments that need to be solved.
Firstly, the issue of navigation for a mobile robot becomes increasingly important in a dynamic environment. This is due to the potential for the robot to encounter unexpected objects in its path that may prevent it from being able to reach its target location. Therefore rendering the mobile robot unable to complete its task.
The second issue is using mobile robotics to sort items in these environments, especially a postal company. This arises from items that require sorting possessing different properties from one another. Thus the robot will need to be able to differentiate items from one another via various properties to be able to sort them.

To overcome the aforementioned issues state of the art robots make use of a wide range of sensing technologies. Such technologies include…

1. Presence sensors: 2D and 3D vision stereo sensors, time of flight (TOF) or light detection and ranging (LIDAR) sensors are used to provide spatial vision sensing and object detection.
2. Force torque sensors: Used in end effectors and grips to monitor movement speed and detect obstacles as well as force being exerted.
3. Environmental sensors: Used to detect and monitor temperature, air quality, humidity, pressure and weight.

To solve the aforementioned problem, I plan to create a solution that utilizes a single mobile robot equipped with a vision sensor, capable of RGB capture, and proximity sensors, in order to detect nearby objects or obstacles.

The solution I produce will be tested in the v-rep simulation software using a scenario that mimics a small industrial environment and will allow for the solutions success to be evaluated. It will require the mobile robot to collect different coloured boxes from three different conveyors and then sort these boxes into their three different groups based on their colour using a vision sensor. [1] The colours and groups will be red, green and blue for simplicity purposes.

2 Details of approach

2.1 Approach

The approach I will take, in order to solve the problem outlined previously and successfully complete the simulation scenario, will use a single mobile robot equipped with a vision sensor capable of capturing RGB video and a number of proximity sensors. The vision sensor will be used to detect the colour of items to enable sorting and the proximity sensors will be used to enable the robot to navigate the environment around it while incurring as few collisions as possible when unexpected objects are encountered.

The collection process for the solution will be the order batching process (OBP). "Order batching is a method of grouping a set of orders into batches for picking multiple orders in a tour in order to minimize the total order picking time” [3, p. 3]. This process was chosen as it aims to minimise the total time spent picking objects in my solution.

The sorting algorithm for the solution will be pick-and-sort (PAS). “After picking all the items, the order picker drops off the picked items at the depot, and also sorts and transports them“ [2, p. 6]. The pick-and-sort process was chosen as it is a strategy used in conjunction with order batch processing to maintain a high pick-rate for pickers [3, p. 2], in this case mobile robots.

Therefore, the combination of these two methods could improve the amount of time required to get an object from a conveyor belt, sort it and have it placed where required in an industrial environment due to the lower total picking time and the higher pick-rate.

2.2 Mobile Robot

The mobile robot chosen was the Kuka YouBot. This was due to two deciding factors.
One factor was the YouBot’s mobility that allowed it to traverse in a multitude of directions due to its four independently controllable wheels. Hence potentially allowing the Kuka YouBot to be more capable of traversing obstacles in its path than its other mobile robot counterparts.
The other deciding factor was its ability to be able to carry multiple objects at a time as a result of its rear platform. This rear platform is very suited to a role that could require multiple objects to be picked up in batches.

However, the Kuka YouBot still required further sensors to be integrated.
Firstly, a vision sensor was added to the YouBot, as seen in Fig. 1. (a). This vision sensor was added to allow the detection of objects colours to enable sorting by colour.

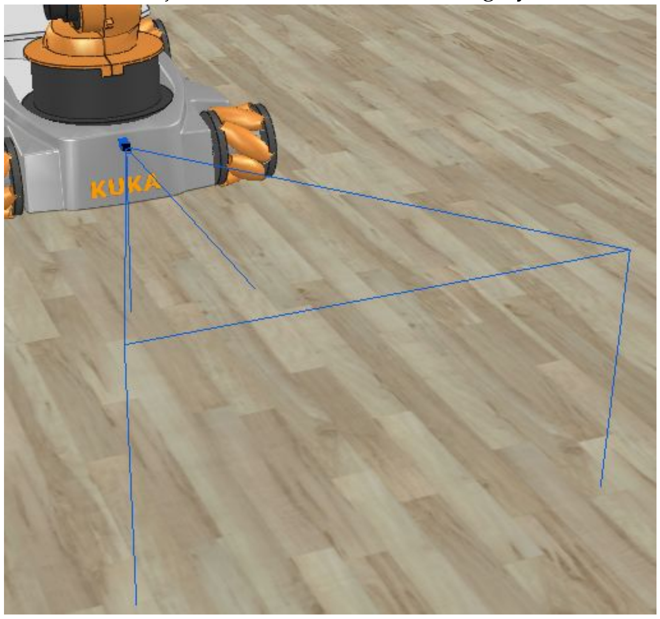


Fig. 1. (a) The vision sensor shown on the front of the Kuka YouBot.

Secondly, four proximity sensors were added to the YouBot, as seen in Fig. 1. (b). The proximity sensors were added to enable the detection of unexpected objects near the robot.

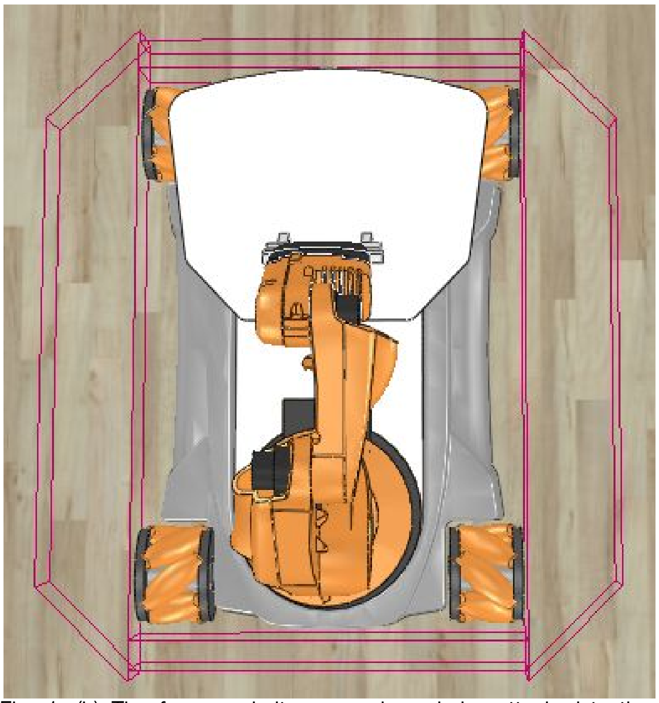


Fig. 1. (b) The four proximity sensor boundaries attached to the Kuka YouBot. A front, rear, left and right sensor are displayed

2.2 Procedure

The procedure needed to provide a solution to the problem outlined in the introduction by picking, sorting and delivering items by colour. While incorporating the methods of OBP and PAS to potentially improve performance with regards to time.

The entire procedure designed for the system can be seen in the flowchart in Fig. 2. This was the procedure implemented as it breaks down each step required of the mobile robot into clear processes and decisions, along with local data storing and accessing. As a result it does not miss a step and could be potentially be applied in multiple different environments and work.

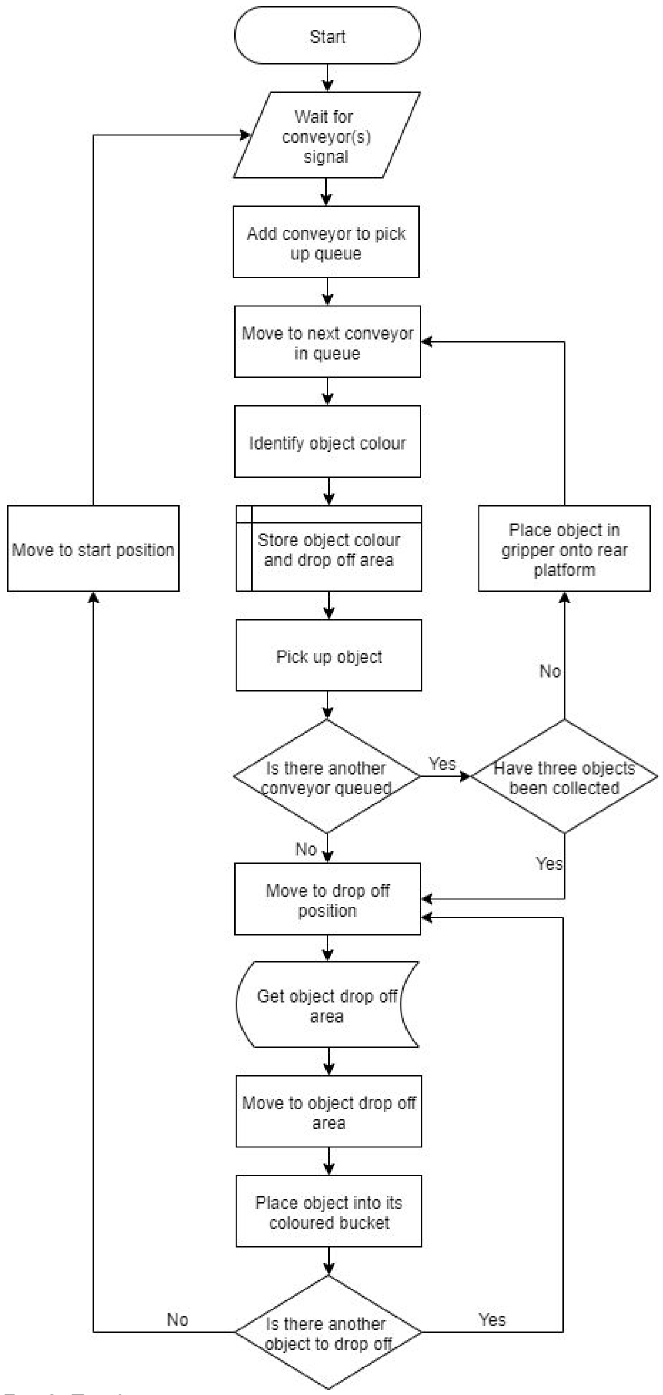

Start
Wait for conveyor(s) signal
Add conveyor to pick up queue
Move to next conveyor in queue
Identify object colour
Store object colour and drop off area
Pick up object
Is there another conveyor queued
Yes
Have three objects been collected
No
Place object in gripper onto rear platform
Yes
No
Move to drop off position
Get object drop off area
Move to object drop off area
Place object into its coloured bucket
Is there another object to drop off
No
Yes
Move to start position

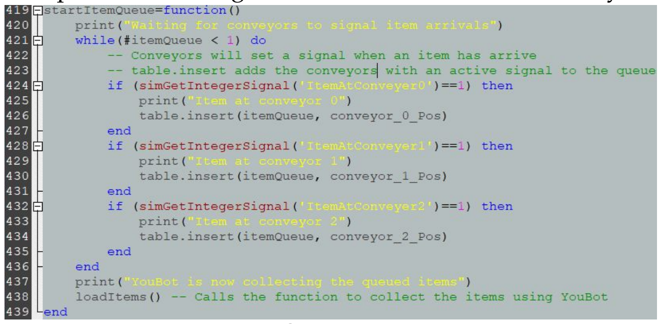
Fig. 2. The flowchart displays the procedure to be implemented into a mobile robot to achieve OBP and PAS.

2.3 Simulation System

When implementing the scenario, as discussed in the introduction, external code was used to give the Kuka YouBot basic functionality and features inside the v-rep simulation software, for all examples of this code see [4].

However, these basic functionalities and features were not enough for the YouBot to be able to carry out the procedure, shown in Fig. 2, in an automated manner. As a result, a number of functions were designed and implemented.

The procedure begins with the startItemQueue() function, Fig. 3. This function has a while loop containing three 'if' statements to check for conveyors with active signals. If a conveyor has an active signal, which means there is an item present, then it gets added to the itemQueue array.



```
startItemQueue=function()
    print("Waiting for conveyors to signal item arrivals")
    while(#itemQueue < 1) do
        -- Conveyors will set a signal when an item has arrive
        -- table.insert adds the conveyors with an active signal to the queue
        if (simGetIntegerSignal('ItemAtConveyer0')==1) then
            print("Item at conveyor 0")
            table.insert(itemQueue, conveyor_0_Pos)
        end
        if (simGetIntegerSignal('ItemAtConveyer1')==1) then
            print("Item at conveyor 1")
            table.insert(itemQueue, conveyor_1_Pos)
        end
        if (simGetIntegerSignal('ItemAtConveyer2')==1) then
            print("Item at conveyor 2")
            table.insert(itemQueue, conveyor_2_Pos)
        end
    end
    print("YouBot is now collecting the queued items")
    loadItems() -- Calls the function to collect the items using YouBot
end
```

Fig. 3. The function startItemQueue is used to 'listen' for conveyors that have active signals and add them to a queue.

For this queueing to be possible, each conveyor has a proximity sensor that when activated by an item sends out a signal unique to that conveyor. A conveyor and its associated code can be seen in Fig. 4 and Fig. 5 respectively.

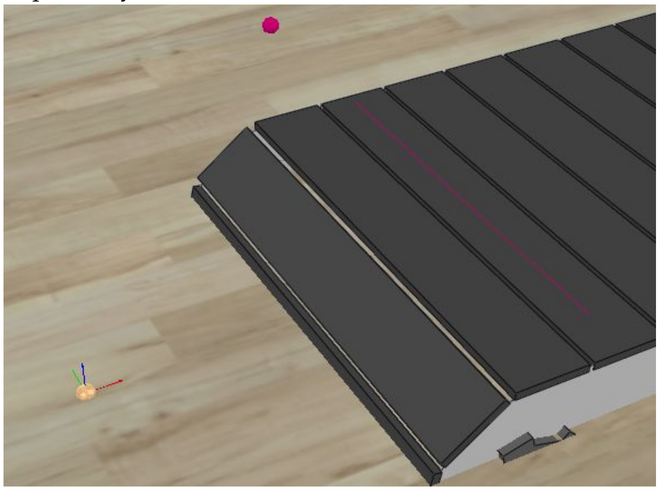
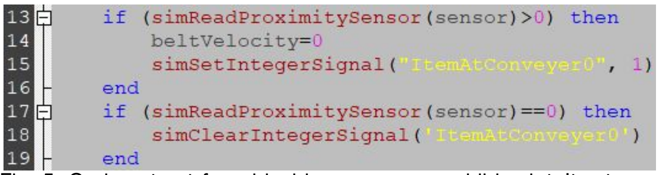


Fig. 4. Each conveyor has a proximity sensor, shown as the red ball with a line, and a dummy placed at its front, shown as the orange ball with xyz axes protruding.



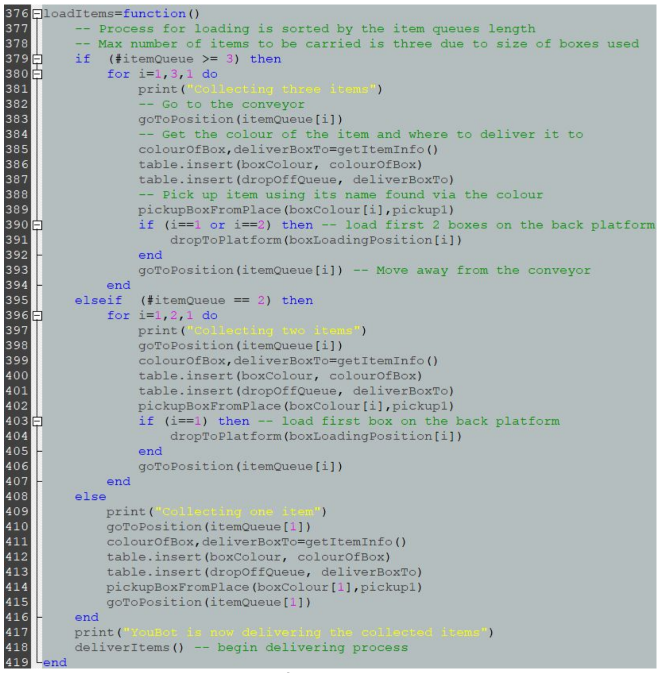
```
    if (simReadProximitySensor(sensor)>0) then
        beltVelocity=0
        simSetIntegerSignal("ItemAtConveyer0", 1)
    end
    if (simReadProximitySensor(sensor)==0) then
        simClearIntegerSignal('ItemAtConveyer0')
    end
```

Fig. 5. Code extract found inside a conveyors childscript. It sets an integer signal to '1' when an item activates the associated proximity sensor. The code also stops the conveyor belt from running.

Once the itemQueue array has a length greater than or equal to one the while loop is broken and the function loadItems() is then called.

The loadItems() function, shown in Fig. 6, is used to collect upto three items in a batch from conveyors that are in the itemQueue array using the YouBot. Depending on the length of the itemQueue array either the 'if', 'elseif' or 'else' statement is executed. The 'if' and 'elseif' statements

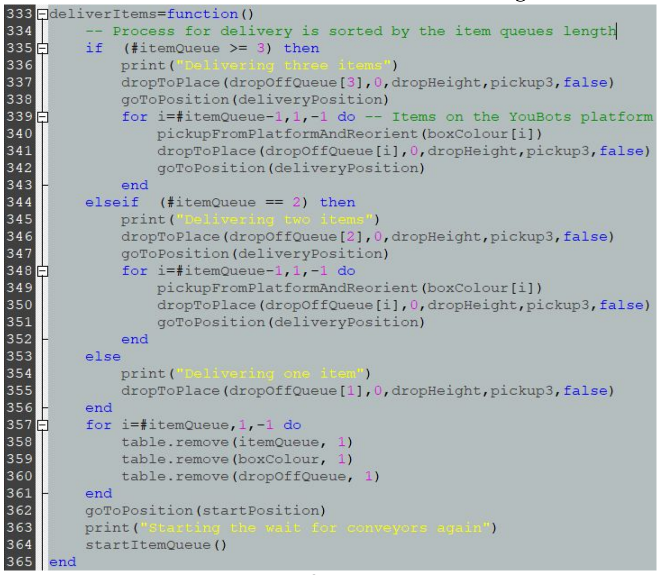
place either two or one item(s) on the YouBot's rear platform and hold an item in the gripper.
After the items in that batch have been collected, the loadItems() function will call the deliverItems() function.



```
loadItems=function()
    -- Process for loading is sorted by the item queues length
    -- Max number of items to be carried is three due to size of boxes used
    if  (#itemQueue >= 3) then
        for i=1,3,1 do
            print("Collecting three items")
            -- Go to the conveyor
            goToPosition(itemQueue[i])
            -- Get the colour of the item and where to deliver it to
            colourOfBox,deliverBoxTo=getItemInfo()
            table.insert(boxColour, colourOfBox)
            table.insert(dropOffQueue, deliverBoxTo)
            -- Pick up item using its name found via the colour
            pickupBoxFromPlace(boxColour[i],pickup1)
            if (i==1 or i==2) then -- load first 2 boxes on the back platform
                dropToPlatform(boxLoadingPosition[i])
            end
            goToPosition(itemQueue[i]) -- Move away from the conveyor
        end
    elseif  (#itemQueue == 2) then
        for i=1,2,1 do
            print("Collecting two items")
            goToPosition(itemQueue[i])
            colourOfBox,deliverBoxTo=getItemInfo()
            table.insert(boxColour, colourOfBox)
            table.insert(dropOffQueue, deliverBoxTo)
            pickupBoxFromPlace(boxColour[i],pickup1)
            if (i==1) then -- load first box on the back platform
                dropToPlatform(boxLoadingPosition[i])
            end
            goToPosition(itemQueue[i])
        end
    else
        print("Collecting one item")
        goToPosition(itemQueue[1])
        colourOfBox,deliverBoxTo=getItemInfo()
        table.insert(boxColour, colourOfBox)
        table.insert(dropOffQueue, deliverBoxTo)
        pickupBoxFromPlace(boxColour[1],pickup1)
        goToPosition(itemQueue[1])
    end
    print("YouBot is now delivering the collected items")
    deliverItems() -- begin delivering process
end
```

Fig. 6. The function loadItems() used to pick items from conveyors in a batch process using the array itemQueue.

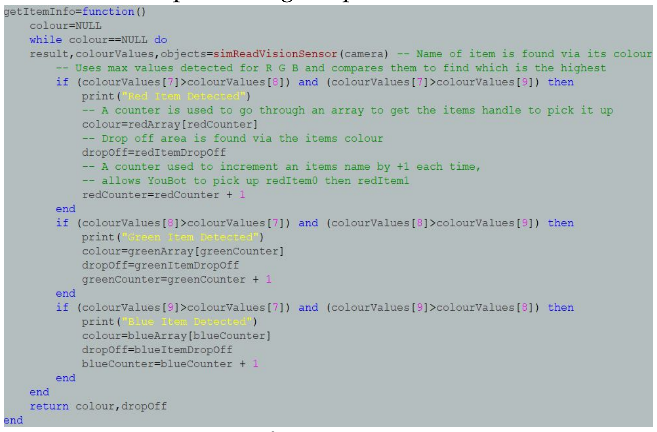
The deliverItems() function, shown in Fig. 7, sorts while delivering items. Depending on the number of items collected in the batch either the 'if', 'elseif' or 'else' statement is executed. Both the 'if' and 'elseif' have extra steps making the YouBot go to the 'deliveryPosition' and then pick up an item off its platform and reorientate it before delivery. The 'else' statement will just deliver the item in the YouBot's gripper. After delivery, information for the now delivered item(s) is deleted from the arrays. The function startItemQueue() is then called again.



```
deliverItems=function()
    -- Process for delivery is sorted by the item queues length
    if  (#itemQueue >= 3) then
        print("Delivering three items")
        dropToPlace(dropOffQueue[3],0,dropHeight,pickup3,false)
        goToPosition(deliveryPosition)
        for i=#itemQueue-1,1,-1 do -- Items on the YouBots platform
            pickupFromPlatformAndReorient(boxColour[i])
            dropToPlace(dropOffQueue[i],0,dropHeight,pickup3,false)
            goToPosition(deliveryPosition)
        end
    elseif  (#itemQueue == 2) then
        print("Delivering two items")
        dropToPlace(dropOffQueue[2],0,dropHeight,pickup3,false)
        goToPosition(deliveryPosition)
        for i=#itemQueue-1,1,-1 do
            pickupFromPlatformAndReorient(boxColour[i])
            dropToPlace(dropOffQueue[i],0,dropHeight,pickup3,false)
            goToPosition(deliveryPosition)
        end
    else
        print("Delivering one item")
        dropToPlace(dropOffQueue[1],0,dropHeight,pickup3,false)
    end
    for i=#itemQueue,1,-1 do
        table.remove(itemQueue, 1)
        table.remove(boxColour, 1)
        table.remove(dropOffQueue, 1)
    end
    goToPosition(startPosition)
    print("Starting the wait for conveyors again")
    startItemQueue()
end
```

Fig. 7. The function deliverItems() is used to deliver items to drop off buckets based on the items colour.

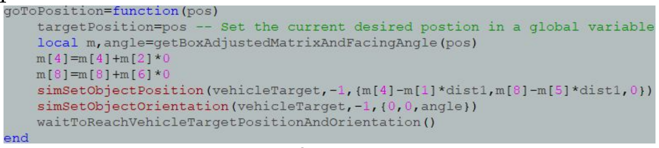
The first supporting function is getItemInfo(), shown in Fig. 8. This function uses the vision sensor on the YouBot to get the items colour using RGB values. The function returns "colour, dropoff", with 'colour' being the items colour and 'dropoff' being the place to deliver the item.



```
getItemInfo=function()
    colour=NULL
    while colour==NULL do
    result,colourValues,objects=simReadVisionSensor(camera) -- Name of item is found via its colour
        -- Uses max values detected for R G B and compares them to find which is the highest
        if (colourValues[7]>colourValues[8]) and (colourValues[7]>colourValues[9]) then
            print("Red Item Detected")
            -- A counter is used to go through an array to get the items handle to pick it up
            colour=redArray[redCounter]
            -- Drop off area is found via the items colour
            dropOff=redItemDropOff
            -- A counter used to increment an items name by +1 each time,
            -- allows YouBot to pick up redItem0 then redItem1
            redCounter=redCounter + 1
        end
        if (colourValues[8]>colourValues[7]) and (colourValues[8]>colourValues[9]) then
            print("Green Item Detected")
            colour=greenArray[greenCounter]
            dropOff=greenItemDropOff
            greenCounter=greenCounter + 1
        end
        if (colourValues[9]>colourValues[7]) and (colourValues[9]>colourValues[8]) then
            print("Blue Item Detected")
            colour=blueArray[blueCounter]
            dropOff=blueItemDropOff
            blueCounter=blueCounter + 1
        end
    end
    return colour,dropOff
end
```

Fig. 8. Function getItemInfo() is used to detect the items colour and assign the place to deliver the item based on its colour.

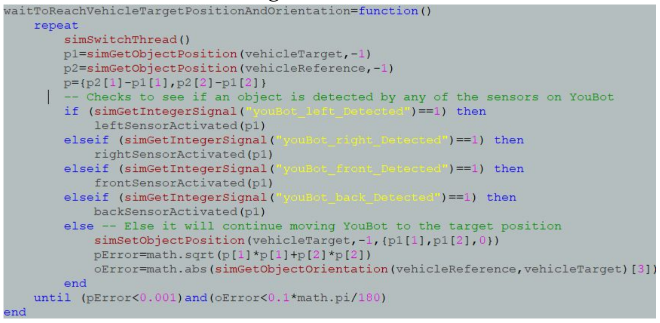
Another supporting function is goToPosition(pos), shown in Fig. 9. It is used to move the YouBot to a specific position in the environment.



```
goToPosition=function(pos)
    targetPosition=pos -- Set the current desired postion in a global variable
    local m,angle=getBoxAdjustedMatrixAndFacingAngle(pos)
    m[4]=m[4]+m[2]*0
    m[8]=m[8]+m[6]*0
    simSetObjectPosition(vehicleTarget,-1,{m[4]-m[1]*dist1,m[8]-m[5]*dist1,0})
    simSetObjectOrientation(vehicleTarget,-1,{0,0,angle})
    waitToReachVehicleTargetPositionAndOrientation()
end
```

Fig. 9. Function goToPosition() accepts a parameter 'pos' and is used to move the Kuka YouBot to a specific position.

The final four supporting functions all serve the same purpose, to prevent the YouBot from colliding with objects detected in its surrounding environment in real time.
These functions are frontSensorActivated(), backSensor Activated(), leftSensorActivated() and rightSensor Activated(). These are only called from within the altered "waitToReachVehicleTargetPositionAndOrientation()" function, as shown in Fig. 10.

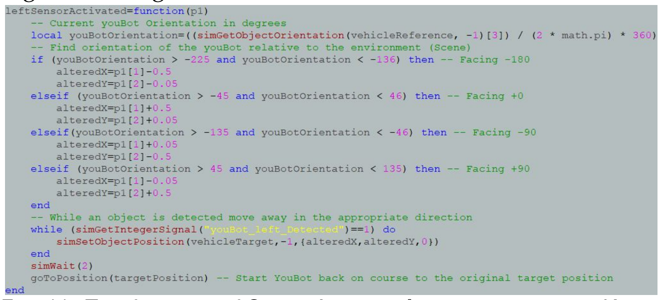


```
waitToReachVehicleTargetPositionAndOrientation=function()
    repeat
        simSwitchThread()
        p1=simGetObjectPosition(vehicleTarget,-1)
        p2=simGetObjectPosition(vehicleReference,-1)
        p={p2[1]-p1[1],p2[2]-p1[2]}
      | -- Checks to see if an object is detected by any of the sensors on YouBot
        if (simGetIntegerSignal("youBot_left_Detected")==1) then
            leftSensorActivated(p1)
        elseif (simGetIntegerSignal("youBot_right_Detected")==1) then
            rightSensorActivated(p1)
        elseif (simGetIntegerSignal("youBot_front_Detected")==1) then
            frontSensorActivated(p1)
        elseif (simGetIntegerSignal("youBot_back_Detected")==1) then
            backSensorActivated(p1)
        else -- Else it will continue moving YouBot to the target position
            simSetObjectPosition(vehicleTarget,-1,{p1[1],p1[2],0})
            pError=math.sqrt(p[1]*p[1]+p[2]*p[2])
            oError=math.abs(simGetObjectOrientation(vehicleReference,vehicleTarget)[3])
        end
    until (pError<0.001)and(oError<0.1*math.pi/180)
end
```

Fig. 10. The altered function waitToReachVehicleTargetPosition AndOrientation().

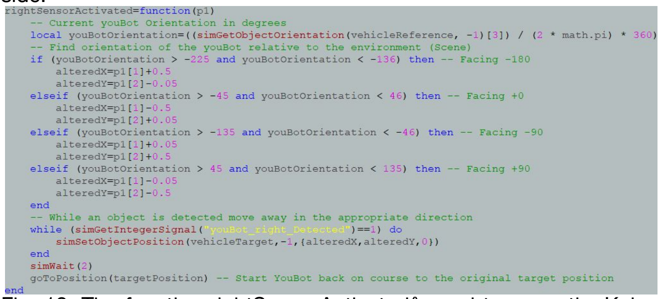
The alterations mean that whilst the YouBot is traversing it will repeatedly check to see if any of the four proximity sensors, as seen in Fig. 1. (b), are active and if so will call one of the four sensor functions.

All four functions are structured the same and take the parameter ‘p1’, ‘p1’ is the current target coordinates for the YouBot. Each function uses an algorithm that gets the orientation of the YouBot to alter the current X and Y target coordinates in order to move the YouBot away from the detected object(s). Thus avoiding collision. All functions finish by calling the goToPosition() function with the original target coordinates. This is all shown in Fig. 11 and Fig. 12.



```
leftSensorActivated=function(p1)
    -- Current youBot Orientation in degrees
    local youBotOrientation=((simGetObjectOrientation(vehicleReference, -1)[3]) / (2 * math.pi) * 360)
    -- Find orientation of the youBot relative to the environment (Scene)
    if (youBotOrientation > -225 and youBotOrientation < -136) then -- Facing -180
        alteredX=p1[1]-0.5
        alteredY=p1[2]-0.05
    elseif (youBotOrientation > -45 and youBotOrientation < 46) then -- Facing +0
        alteredX=p1[1]+0.5
        alteredY=p1[2]+0.05
    elseif(youBotOrientation > -135 and youBotOrientation < -46) then -- Facing -90
        alteredX=p1[1]+0.05
        alteredY=p1[2]-0.5
    elseif (youBotOrientation > 45 and youBotOrientation < 135) then -- Facing +90
        alteredX=p1[1]-0.05
        alteredY=p1[2]+0.5
    end
    -- While an object is detected move away in the appropriate direction
    while (simGetIntegerSignal("youBot_left_Detected")==1) do
        simSetObjectPosition(vehicleTarget,-1,{alteredX,alteredY,0})
    end
    simWait(2)
    goToPosition(targetPosition) -- Start YouBot back on course to the original target position
end
```

Fig. 11. The function leftSensorActivated() used to move the Kuka YouBot away from an object it could collide with on its left hand side.



```
rightSensorActivated=function(p1)
    -- Current youBot Orientation in degrees
    local youBotOrientation=((simGetObjectOrientation(vehicleReference, -1)[3]) / (2 * math.pi) * 360)
    -- Find orientation of the youBot relative to the environment (Scene)
    if (youBotOrientation > -225 and youBotOrientation < -136) then -- Facing -180
        alteredX=p1[1]+0.5
        alteredY=p1[2]-0.05
    elseif (youBotOrientation > -45 and youBotOrientation < 46) then -- Facing +0
        alteredX=p1[1]-0.5
        alteredY=p1[2]+0.05
    elseif (youBotOrientation > -135 and youBotOrientation < -46) then -- Facing -90
        alteredX=p1[1]+0.05
        alteredY=p1[2]+0.5
    elseif (youBotOrientation > 45 and youBotOrientation < 135) then -- Facing +90
        alteredX=p1[1]-0.05
        alteredY=p1[2]-0.5
    end
    -- While an object is detected move away in the appropriate direction
    while (simGetIntegerSignal("youBot_right_Detected")==1) do
        simSetObjectPosition(vehicleTarget,-1,{alteredX,alteredY,0})
    end
    simWait(2)
    goToPosition(targetPosition) -- Start YouBot back on course to the original target position
end
```

Fig. 12. The function rightSensorActivated() used to move the Kuka YouBot away from an object it could collide with on its right hand side.

3 Results

The procedure, shown in Fig. 2, was developed into a system inside of the v-rep simulation software. The scenario used for testing required the YouBot to collect six coloured boxes from three different conveyors in various batch sizes and then sort the boxes by delivering them to the correctly coloured bucket.

The criteria for a simulation run to be considered a success is as follows…

1. All coloured boxes must be collected from the conveyors in batch sizes of 1, 2 or 3.
2. All coloured boxes must be put in the correct coloured bucket.
3. The YouBot must experience two or less collisions.
4. The YouBot must not experience a collision that prevents it from being able to complete the scenario.

The YouBot, with its added sensors, paired with the procedure designed was able to successfully complete the scenario every time and meet all the criteria. However, there were some inefficiencies, mainly in the YouBot’s movement and pathing, that could be improved upon to make the overall process take less time per item to be picked, sorted and delivered than it currently does.

In the simulation the YouBot picked up the boxes in the batch sizes of 1, 3 and then 2, while sorting and delivering each batch before starting the next as expected.

4 Discussions and conclusions

In conclusion, the simulation scenario used may have been too simple to test the procedure and mobile robot designed and implemented extensively. However, I am confident the procedure designed would be able to handle a larger more complex simulation scenario because of how robust it is.

Despite this there were issues encountered that could be developed further to improve the overall solution.

Firstly, the implemented method of using proximity sensors to enable the detection of objects or obstacles around the YouBot, Fig. 1. (b), and redirect it to avoid collision, although it did not fail, in practice was very slow to react and resulted in near collisions, due to it being reactive rather than proactive. Therefore this method could be improved by instead using 3d vision stereo sensors to provide spatial vision and proactively detect objects using depth data. This data could then be used when calculating the path of the YouBot in real time to avoid collisions.

Another issue was the movement of the YouBot. The method used to calculate the path for the YouBot to follow always selected the shortest path. This results in the YouBot often attempting to move diagonally through objects, like conveyors. To improve this, the method for movement could be overhauled in coordination with the use of 3d vision stereo sensors to enable the YouBot to detect objects and then calculate the shortest path without obstacles to the target. This would also improve the efficiency in regards to time taken as a result of the distance travelled.

Lastly, the solution could be further improved if when sorting and delivering items that are the same the YouBot identifies this and delivers them at the same time, one after another. Instead of delivering one and going to the ‘deliveryPosition’ then delivering the other. This would make the process more time efficient.

References

[1] C. Liang, K.J. Chee, Y. Zou, H. Zhu, A. Causo, S. Vidas, T. Teng, I.M. Chen, K.H. Low, and C.C. Cheah. “Automated Robot Picking System for E-Commerce Fulfillment Warehouse Application”, [Online], Taipei, Taiwan, 2015.

[2] Gong, Yeming, Winands, Erik M. M., de Koster, Rene B. M., "A Real-Time Picking and Sorting System in E-Commerce Distribution Centers", [Online], Georgia Southern Univ, 2010. Available:https://pdfs.semanticscholar.org/2498/c41d0774ae07847d5bf3b05ece3ea5b64066.pdf

[3] X, Jiang. Y, Zhou. Y, Zhang. L, Sun. X, Hu. “Order batching and sequencing problem under the pick-and-sort strategy in online supermarkets”, [Online], in Conf. Knowledge-Based and Intelligent Information & Engineering Systems, Belgrade, Serbia, 2018. Available: https://www.sciencedirect.com/science/article/pii/S1877050918312304

[4] Z, Ju. “Using Kuka YouBot”, [Online]. Available: https://moodle.port.ac.uk/pluginfile.php/1609355/mod_resource/content/0/USING%20KUKA%20YOUBOT-Vrep.pptx