A. Possible Working Algorithms

1. Temperature Sensor

Initialize the LCD {Monitor the temperature from the temperature sensor

{If (temp<set value) {check the water content

{If (water content<set) switch ON the motor and send the info. to smart phone ELSE

{Motor is OFF}}}

2. Proximity Sensing

X[i – 1] = X[0]

l[i – 1] = 0

loop 1:

D[i] = X[i] – X[i – 1]

Is (ABS (D[i]) greater than DT)?

true: l[i – 1] + D[i]

else: l[i] = l[i – 1]

Is (l[i] ≥ IT)

true:

Object detected

l[i – 1] = l[i]

else:

Object not detected

l[i – 1] = l[i] * L

a. Parameters

IT = Integration threshold

DT = Derivative threshold

L = Leakage factor

X[i] = current sample point

X[i – 1] = previous sample point

D[i] = Derivative

l[i] = Integral of Derivative

l[i – 1] = Previous Integral of Derivative

3. Ultrasonic Distance Sensor

//Hookup HC-SR04 with Trig to Arduino Pin10, Echo to Arduino pin13

//Maximum Distance is [---] cm

```
#define TRIGGER_PIN10

#define ECHO_PIN13

#MAX_DISTANCE [ --- ]


New Ping Sonar (TRIGGER_PIN, ECHO_PIN,

MAX_DISTANCE);

float duration, distance;

void setup (   ){

        serial begin (9600);

}

void loop (   ){

        duration = sonar.ping (   );


#Determine distance from duration

#Use 343 metres per second as speed of sound


distance = (duration/2)*0.0343;


//send results to Serial Monitor
```

Serial.print (“Distance = ”):

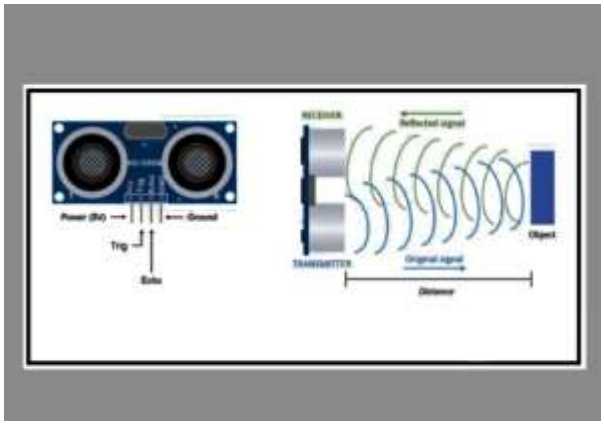


Figure 4A. Ultrasonic Distance Sensor

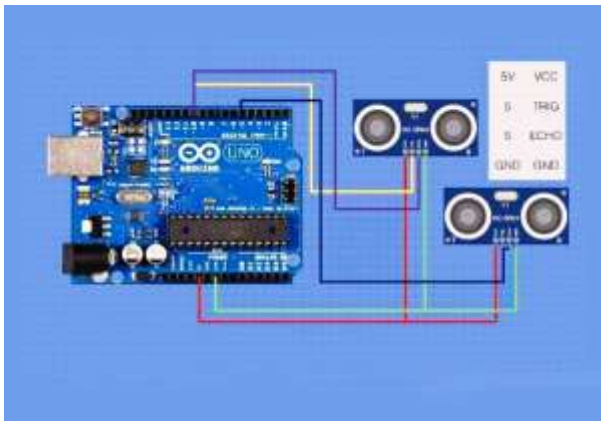


Figure 4B. Architecture of Ultrasonic Distance Sensor

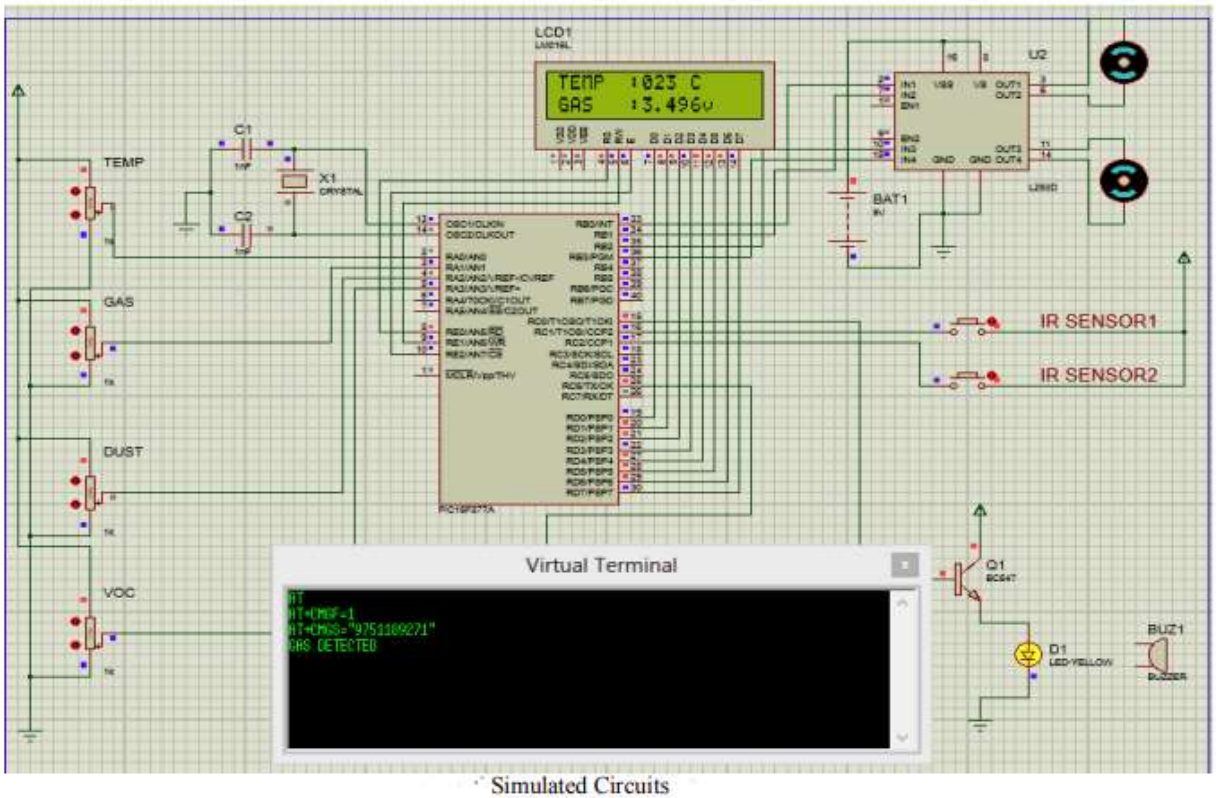
a. Parameters

VCC – 5 volt power connection

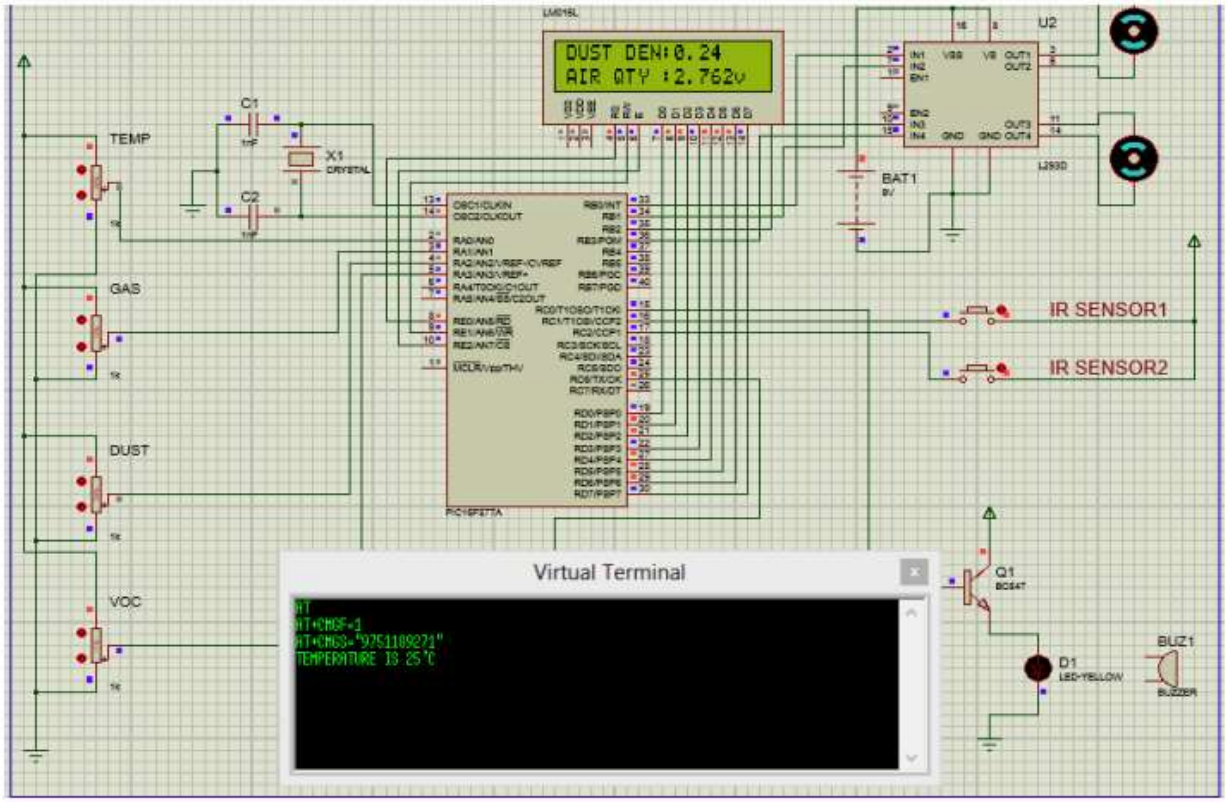
TRIG – Trigger pin (input)

ECHO – Echo pin (output)

GND – Ground



Simulated Circuits



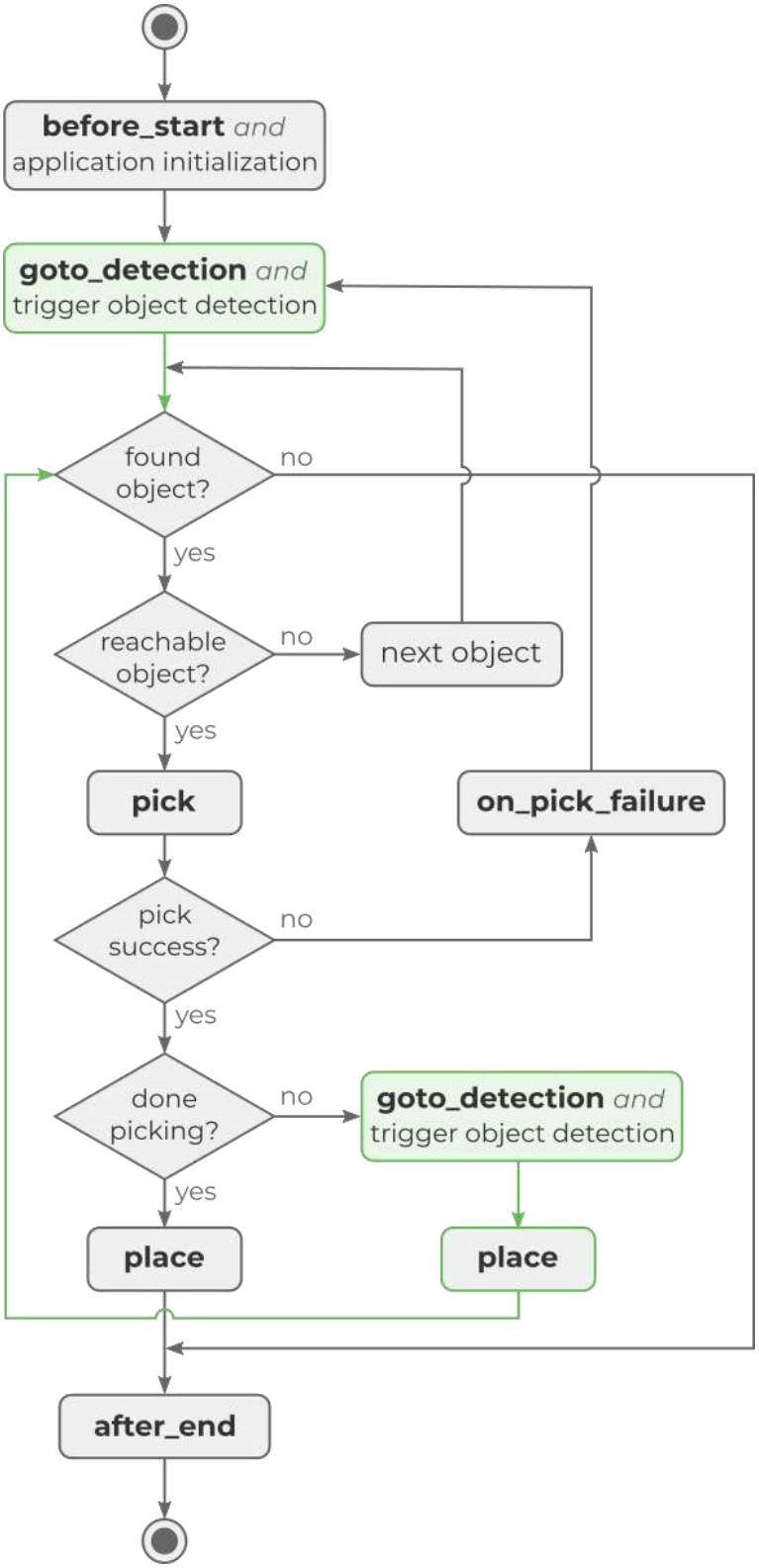
Simulated Circuits

Pick and Place function

```
1  def pick_and_place(setup, product, target_picks=-1, retries=5):
2  if not pickit_is_running():
3      print("Pickit is not in robot mode. Please enable it in the web interface.")
4      halt()
5
6  before_start()
7  pickit_configure(setup, product)
8
9  goto_detection()
10  pickit_find_objects_with_retries(retries)
11  pickit_get_result()
12
13  picks = 0
14  while True:
15      if not pickit_object_found():
16          # There are no pickable objects, bail out.
17          break
18
19      # Compute bin_entry, pre-pick, post-pick and bin_exit points.
20      compute_extraction_path()
21
22      if pickit_is_reachable(PickitPick, BinEntry, PrePick, PostPick, BinExit):
23          # Object is pickable! Attempt pick.
24          pick_success = pick()
25          if pick_success:
26              picks += 1
27              done_picking = target_picks > 0 and picks == target_picks
28              if done_picking:
29                  # Target picks reached. Place without further detections.
30                  place()
31                  break
32              else:
33                  # Target picks not reached. Place and detect next object.
34                  goto_detection()
35                  pickit_find_objects_with_retries(retries)
36                  place()  # In parallel to detection, saves cycle time.
37                  pickit_get_result()
```

```
38          else:
39              # Picking failed, skip place and detect next object.
40              on_pick_failure()
41              goto_detection()
42              pickit_find_objects_with_retries(retries)
43              pickit_get_result()
44      else:
45          # Object is unreachable, get the next detection, if any.
46          pickit_next_object()
47          pickit_get_result()
48
49  after_end()
50
51  return picks
```

Flowchart


before_start and application initialization
goto_detection and trigger object detection
found object?
no
yes
reachable object?
no
next object
yes
pick
on_pick_failure
pick success?
no
yes
done picking?
no
goto_detection and trigger object detection
yes
place
place
after_end

The lines that differ with respect to the simple pick and place logic are highlighted above, and implement the following additional features:

- The ability to not only *pick all objects* (the default), but also to specify a *target number of picks*.
- Some robot tools have the means to measure pick success. When this is available, cycle time can be optimized by skipping the place motion on pick failure, and instead proceed to trigger a new object detection.
- Wrap the logic in a `pick_and_place` function, so it can be reused more easily and with less code duplication. It outputs the *number of successful picks*, and takes as inputs:
  - **Required:** the Pickit configuration, `setup` and `product`.
  - **Optional:** `target_picks`, which defaults to -1 (*pick all*); and `retries`, which denotes how many times to retry object detection when the Region of Interest (ROI) is not empty, but no objects are detected.

Application-specific hooks

The `pick_and_place` function requires the following application-specific hooks to be defined. Click on the entries below to expand them and learn more about their default implementation and behavior:

before_start

```
# Action performed once before starting pick and place.
def before_start():
    gripper_release()
```

This hook is executed *once* before starting to pick and place. It's recommended to add logic required to bring the robot to a sane configuration before starting to pick objects. This can be especially useful when the robot program was previously interrupted at mid-run.

By default, it makes sure the gripper is open to prepare it for picking an object.

goto_detection

```
# Move the robot to the point from which object detection is triggered.
def goto_detection():
  movej(Detect)
```

This is a motion sequence that moves the robot to the point from which object detection is triggered. For simple applications, this typically corresponds to a single waypoint.

Some applications using a robot-mounted camera might require a non-constant `Detect` point. For instance, when a bin is wider than the camera field of view, multiple detection points are required to fully cover it.

compute_extraction_path

When picking from shallow bins, or in non-bin picking applications, it's typically sufficient to use a three-point extraction path `PrePick` → `PickitPick` → `PostPick`.

```
# Compute pre-pick and post-pick points.
def compute_extraction_path(PickitPick, pre_pick_offset=100,
post_pick_offset=100):
  PrePick = PickitPick * Pose(0, 0, -pre_pick_offset, 0, 0, 0)

  PostPick = PickitPick
  PostPick.z = PostPick.z + post_pick_offset
```

More precisely the `PrePick` is chosen such that the approach motion is aligned with the object, while `PostPick` is chosen for a straight-up retreat (which makes it less likely to collide with obstacles like the bin).

When picking from deep bins, a five-point extraction path is preferred, which is similar to the above, but with the addition of `BinEntry` at the beginning of the path

and `BinExit` at the end. These points are vertical translations of `PrePick` and `PostPick`, respectively, as shown below.

```
# Compute BinEntry, PrePick, PostPick and BinExit points.
def compute_extraction_path(PickitPick, bin_entry_z=300, pre_pick_offset=100,
                            post_pick_offset=100, bin_exit_z=300):
    PrePick = PickitPick * Pose(0, 0, -pre_pick_offset, 0, 0, 0)

    PostPick = PickitPick * Pose(0, 0, -post_pick_offset, 0, 0, 0)

    BinEntry = PrePick
    BinEntry.z = bin_entry_z

    BinExit = PostPick
    BinExit.z = bin_exit_z
```

The first path segment performs bin entry in a safe way: It brings the tool to `BinEntry` and descends vertically to `PrePick`. The same concept is applied to the last path segment (`PostPick` to `BinExit`), which exits the bin. The parameters `bin_entry_z` and `bin_exit_z` are user-defined, and should always be larger than the bin height.

The user can decide which path to use, and then modify the other parts of the program accordingly. In particular, it will affect the argument of the function `pickit_is_reachable()` and the move commands inside `pick()`.

pick

```
# Sequence for performing the picking motion:
# - Starts and ends at AbovePickArea, a point reachable in a collision-free way.
# - BinEntry --> PrePick: Linear move along the Z direction to enter in the bin.
# - PrePick --> PickitPick: Linear approach to the pick point.
# - A grasping action.
# - PickitPick --> PostPick: Linear retreat away from the pick point.
# - PostPick --> BinExit: Linear move along the Z direction to exit the bin.
def pick():
```

```
movej(AbovePickArea)
movel(BinEntry)
movel(PrePick)
movel(PickitPick)
gripper_grasp()  # For a suction-like gripper, do this one line above.
movel(PostPick)
movel(BinExit)
movel(AbovePickArea)

return gripper_pick_success()
```

Note

`movej(p)` and `movel(p)` represent a robot motion to reach waypoint `p` following a path interpolated linearly in joint or Cartesian space, respectively. Motions between points of the pick sequence should be *linear* (`movel(p)`) to guarantee a predictable path. *Joint motions* (`movej(p)`) are discouraged during the pick sequence, as the robot may take an unexpected path that causes a collision with the bin (if present) or neighboring parts. They are however recommended for motions in open, unconstrained space, such as in the `goto_detection` and `place` sequences.

The pick sequence performs the actual picking motion, which consists of a linear approach to the pick point, a grasping action, and a linear retreat away from it.

- The sequence starts and ends at `AbovePickArea`, a waypoint known to be reachable without collision both from the pick area and from the other user-defined waypoints.
- The place where `gripper_grasp()` is called depends on the type of gripper. Fingered grippers perform the grasp action at the pick point, but for suction-like grippers this typically takes place *before* heading to the pick point.
- The pick point, `PickitPick`, is computed by Pickit.
- Points `PrePick` and `PostPick` (plus optionally `BinEntry` and `BinExit`) are computed in compute_extraction_path().
- The `pick()` function returns a boolean indicating whether the pick was successful or not.

Note

The check represented by `gripper_pick_success()` assumes that the gripper has the means to check pick success from sensor input (like vacuum or force). If this is not the case for your gripper, the `pick()` function can simply `return True` always, and the pick failure logic will never be triggered.

on_pick_failure

```
# Action taken when picking an object failed.
def on_pick_failure():
    gripper_release()
```

This hook is executed whenever the pick success check fails, (see `gripper_check_success()` in the pick hook). The default implementation opens the gripper to prepare it for picking the next object.

place

```
# Sequence for placing the object at the specified dropof location.
def place():
    movej(Dropoff)
    gripper_release()
```

This sequence places the object at the specified dropoff location. For simple applications the implementation is trivial, as shown above. However, some applications require more advanced place motions.

The robot sometimes needs to know about the way the object was picked, in order to place it appropriately. Refer to the smart placing examples to learn how to do this with minimal programming effort.

It can also be the case that the drop-off point is not constant, as when parts need to be stacked or palletized. Many robot programming languages provide helpers and templates for stacking and palletizing, which can replace the fixed `Dropoff` point.

after_end

```
# Action performed once after pick and place has finished.
def after_end():
    if not pickit_object_found():
        if pickit_empty_roi():
            print("The ROI is empty.")
        elif pickit_no_image_captured():
            print("Failed to capture a camera image.")
        else:
            print("The ROI is not empty, but the requested object was not found or is unreachable.")
```

This hook is executed *once* after pick and place has finished. The proposed implementation identifies the termination reason and prints an informative statement if there are no more pickable objects. This is very useful to debug your application while you're setting it up.

When getting your application ready for production, you should handle the cases that make sense to you with appropriate logic. For instance, a continuous-running application might want to request more parts once all pickable objects have been processed.

```
def after_end():
    # Unrecoverable error. Raise alarm and stop program.
    if pickit_no_image_captured():
        alarm("Failed to capture a camera image.")
        halt()

    if not pickit_empty_roi():
        # Save a snapshot to learn why no objects were detected in a non-empty ROI.
        pickit_save_snapshot()

    # Request more parts to start picking all over again.
```

```
    feed_more_parts()
```

Notice how the application only stops on non-recoverable errors, and triggers saving a snapshot whenever it fails to empty the ROI. Inspecting these snapshots allow to improve the application by answering questions like:

- Are there actually unpicked objects in the ROI, or are there unexpected contents in it?
- If there are objects, are they detected but unpickable by the robot (because they are unreachable or the picking action failed)?
- If there are objects, but they are not detected, can we optimize the detection parameters or camera location to make them detectable?

FOR USAGE

```
# Application inputs (needs replacing with actual values).
Detect = [x,y,z,rx,ry,rz]
AbovePickArea = [x,y,z,rx,ry,rz]
Dropoff = [x,y,z,rx,ry,rz]

setup = 1
product = 1

def gripper_release():
    # Add custom gripper release logic.

def gripper_grasp():
    # Add custom gripper grasp logic.

def gripper_pick_success():
    # Add custom gripper pick success check. Returns a boolean.
    # If you don't have the means to measure pick success, return always True.

# Pick all objects and write number of successful picks to 'picks'.
picks = pick_and_place(setup, product)
```

Robot-mounted camera

If a robot-mounted camera is used, it's not possible to perform multiple detection retries (including camera captures) in parallel to the place motion sequence, as camera capture can only take place from the Detect point. To correctly handle the robot-mounted camera scenario, replace lines 31-34 with the following:

```
# Try first a single detection in parallel to place...
goto_detection()
capture_ok = pickit_capture_image()
if capture_ok:
    pickit_process_image()
    place()
    pickit_get_result()
# If not successful, detect with retries. No longer in parallel with motions.
if not capture_ok or not pickit_object_found():
    goto_detection()
    pickit_find_objects_with_retries(retries)
    pickit_get_result()
```

Notice the use of the functions pickit_capture_image() and pickit_process_image().

Collision recovery

During the pick sequence, unexpected collisions may occasionally happen, which, if unhandled, can trigger a protective stop.

To prevent robot downtime and the human intervention required to recover from a protective stop, it is recommended to add a collision recovery routine to the robot program, if supported by the robot programming language. An example recovery strategy would be to release the gripper (and picked part, if any), move safely to the Detect point to trigger a new detection, and pick a new part, as shown below:

```
# Sequence for performing the picking motion
def pick():
    movej(AbovePickArea)
    movel(BinEntry)
    movel(PrePick)
    movel(PickitPick)
    gripper_grasp()  # For a suction-like gripper, do this one line above.
    movel(PostPick)
    movel(BinExit)
    movel(AbovePickArea)

    Error # Catch error in this loop.
        if collision:   # Collision detected by robot, trying to recover.
            # Stop motion and clear path, if needed by the programming language.
            stop_motion()
            # Raise the gripper.
            movel(BinExit)
            return false;

    return gripper_pick_success()
```

Warning

If another collision is detected while executing the collision recovery sequence, it will not be handled and a protective stop will be raised.

The above sequence does not detect collisions outside the bin. If desired, other collision recovery routines can be added to handle such cases.

Multi-pose calibration

Fully automated

```
if not pickit_is_running():
    print("Pickit is not in robot mode. Please enable it in the web interface.")
    halt()

# Calibration poses (needs replacing with actual values).
calib_pose_1  = [x,y,z,rx,ry,rz]
calib_pose_2  = [x,y,z,rx,ry,rz]
calib_pose_3  = [x,y,z,rx,ry,rz]
calib_pose_4  = [x,y,z,rx,ry,rz]
calib_pose_5  = [x,y,z,rx,ry,rz]
calib_pose_6  = [x,y,z,rx,ry,rz]
calib_pose_7  = [x,y,z,rx,ry,rz]
calib_pose_8  = [x,y,z,rx,ry,rz]
calib_pose_9  = [x,y,z,rx,ry,rz]
calib_pose_10 = [x,y,z,rx,ry,rz]

# Configure calibration using current calibration settings in the UI.
# You can alternatively be explicit and force the configuration settings like so:
#
# config_ok = pickit_configure_calibration(METHOD_MULTI_POSE, CAMERA_MOUNT_ON_ROBOT)

config_ok = pickit_configure_calibration()

if not config_ok:
    # Add actions to perform on failed configuration.
    exit()  # Do not continue program execution.

# Move to each calibration pose and trigger a calibration plate detection.
movej(calib_pose_1)
pickit_find_calibration_plate()

movej(calib_pose_2)
pickit_find_calibration_plate()
```

```
34
35movej(calib_pose_3)
36pickit_find_calibration_plate()
37
38movej(calib_pose_4)
39pickit_find_calibration_plate()
40
41movej(calib_pose_5)
42pickit_find_calibration_plate()
43
44movej(calib_pose_6)
45pickit_find_calibration_plate()
46
47movej(calib_pose_7)
48pickit_find_calibration_plate()
49
50movej(calib_pose_8)
51pickit_find_calibration_plate()
52
53movej(calib_pose_9)
54pickit_find_calibration_plate()
55
56movej(calib_pose_10)
57pickit_find_calibration_plate()
58
59# Compute the robot-camera calibration.
60calibration_ok = pickit_compute_calibration()
61
62if not calibration_ok:
63    # Add actions to perform on failed calibration.
```

Camera-Robot Calibration

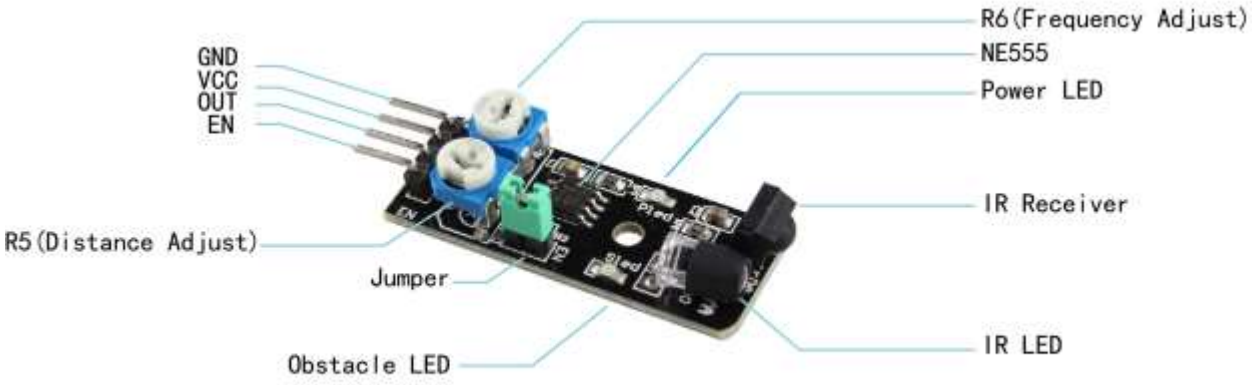
File Structure

```
./dataset
+-- dataset1
|  +-- out.json
```

```
| +-- cam_0
| | +-- images
| | | +-- 000.png
| | | +-- ...
| | +-- meta
| | | +-- 000.json
| | | +-- ...
| | +-- table*
| | | +-- images
| | | | +-- ...
| | | +-- meta
| | | | +-- ...
| +-- cam_1
...
```

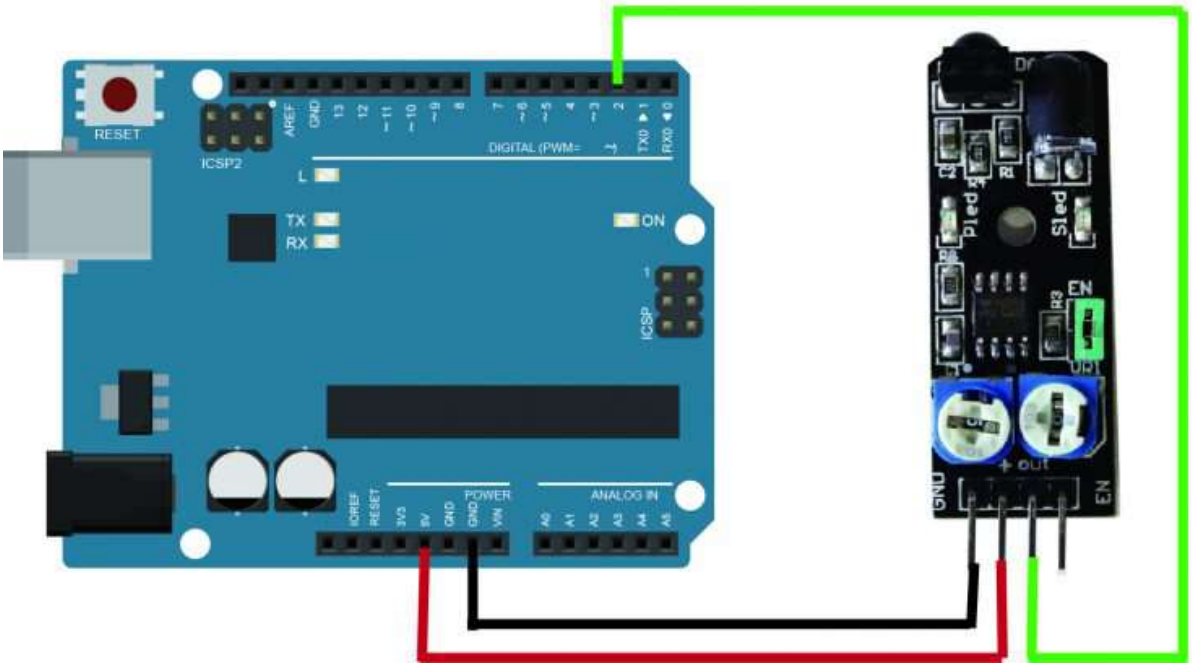
The calibration script scans the sub-directories in a given dataset, each sub-directory contains the sample associated with one camera, including a images and a meta folder. The *.json file for each sample has the joint angle of the manipulator and corresponding image name. The table folder is optional, it uses the calibration parameter associated with the directory it is in to estimate the table surface height w.r.t the manipulator's base frame.

IR Obstacle Avoidance Sensor



CONNECTION

Make sure the **Enable Jumper** is placed on the Infrared **Obstacle Avoidance sensor,** Build the circuit as below digram:



CODE PROGRAM

After above operations are completed, connect the board to your computer using the USB cable. The green power LED (labelled **PWR**) should go on.Open the Arduino IDE and choose corresponding board type and port type for you project. Then load up the following sketch onto your board.

```
int LED = 13; // Use the onboard Uno LED
int isObstaclePin = 2;  // This is our input pin
int isObstacle = HIGH;  // HIGH MEANS NO OBSTACLE

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(isObstaclePin, INPUT);
  Serial.begin(9600);
```

```
}

void loop() {
  isObstacle = digitalRead(isObstaclePin);
  if (isObstacle == LOW)
  {
    Serial.println("OBSTACLE!!, OBSTACLE!!");
    digitalWrite(LED, HIGH);
  }
  else
  {
    Serial.println("clear");
    digitalWrite(LED, LOW);
  }
  delay(200);
}
```

LPG SENSOR

```
// code for with led only

 int LED = 12;

 int LPG_sensor = 3;// MQ-6 SENSOR

 int LPG_detected;

 void setup()

 {

 Serial.begin(9600);

 pinMode(LED, OUTPUT);

 pinMode(LPG_sensor, INPUT);
```

```
}
void loop()
{
LPG_detected = digitalRead(ALCOHOL_sensor);
Serial.println(LPG_detected);
if (ALCOHOL_detected == 1)
{
Serial.println("LPG detected...");
digitalWrite(LED, HIGH);
}
else
{
Serial.println("No LPG detected ");
digitalWrite(LED, LOW);
}
}
```

DUST SENSOR

```
int dustPin = 0; // dust sensor - Arduino A0 pin
int ledPin = 2;

float voltsMeasured = 0;
float calcVoltage = 0;
float dustDensity = 0;

void setup()
{
  Serial.begin(57600);
  pinMode(ledPin,OUTPUT);
}


void loop()
{
  digitalWrite(ledPin,LOW); // power on the LED
  delayMicroseconds(280);

  voltsMeasured = analogRead(dustPin); // read the dust value
```

```
  delayMicroseconds(40);
  digitalWrite(ledPin,HIGH); // turn the LED off
  delayMicroseconds(9680);

  //measure your 5v and change below
  calcVoltage = voltsMeasured * (5.0 / 1024.0);
  dustDensity = 0.17 * calcVoltage - 0.1;
  Serial.println("GP2Y1010AU0F readings");
  Serial.print("Raw Signal Value = ");
  Serial.println(voltsMeasured);
  Serial.print("Voltage = ");
  Serial.println(calcVoltage);
  Serial.print("Dust Density = ");
  Serial.println(dustDensity); // mg/m3
  Serial.println("");
  delay(1000);
}
```