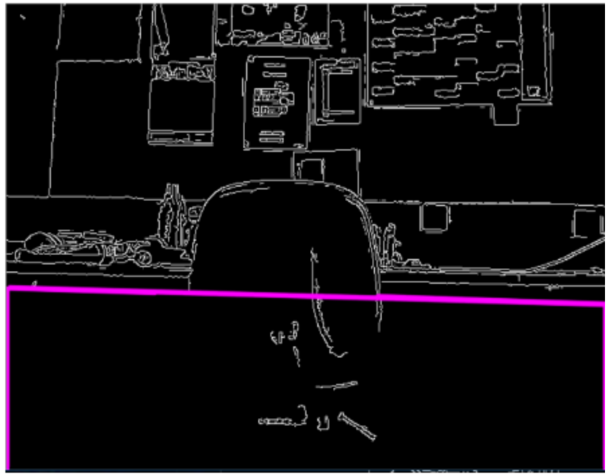
Andy Everitt

Assisted control over wheelchair movement to aid users in approaching desks/tables.

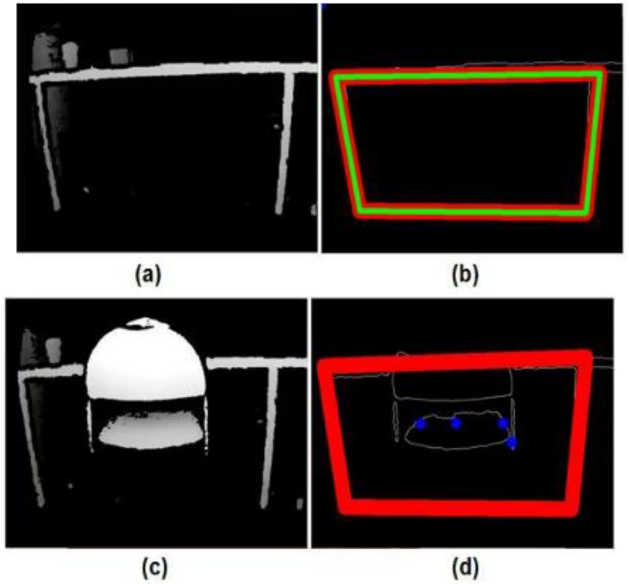
Wheelchair Docking

Background

- E. Langdon
  - R200 colour image
  - No obstacle detection
  - Limited reliablility
- L. Martins
  - R200 depth image
  - Only detects maximum of one obstacle
  - Requires each corner of the desk to be in each quadrant on image



E. Langdon [1]

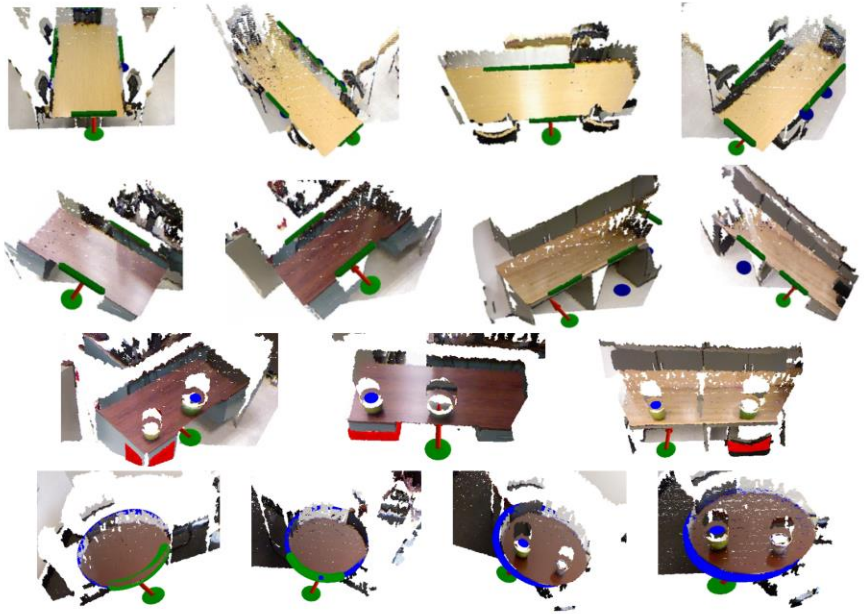


L. Martins [2]

[1] Elliot Langdon and Richard Green, ‘Recognition of Docking Locations for Electric Wheelchairs’, University of Canterbury, Computer Vision Lab, 2017.
[2] Lucas Amilton Martins and Richard Green, ‘Automated Recognition of Docking Locations for Electric-Powered Wheelchairs’, University of Canterbury, Computer Vision Lab, 2016.

Background

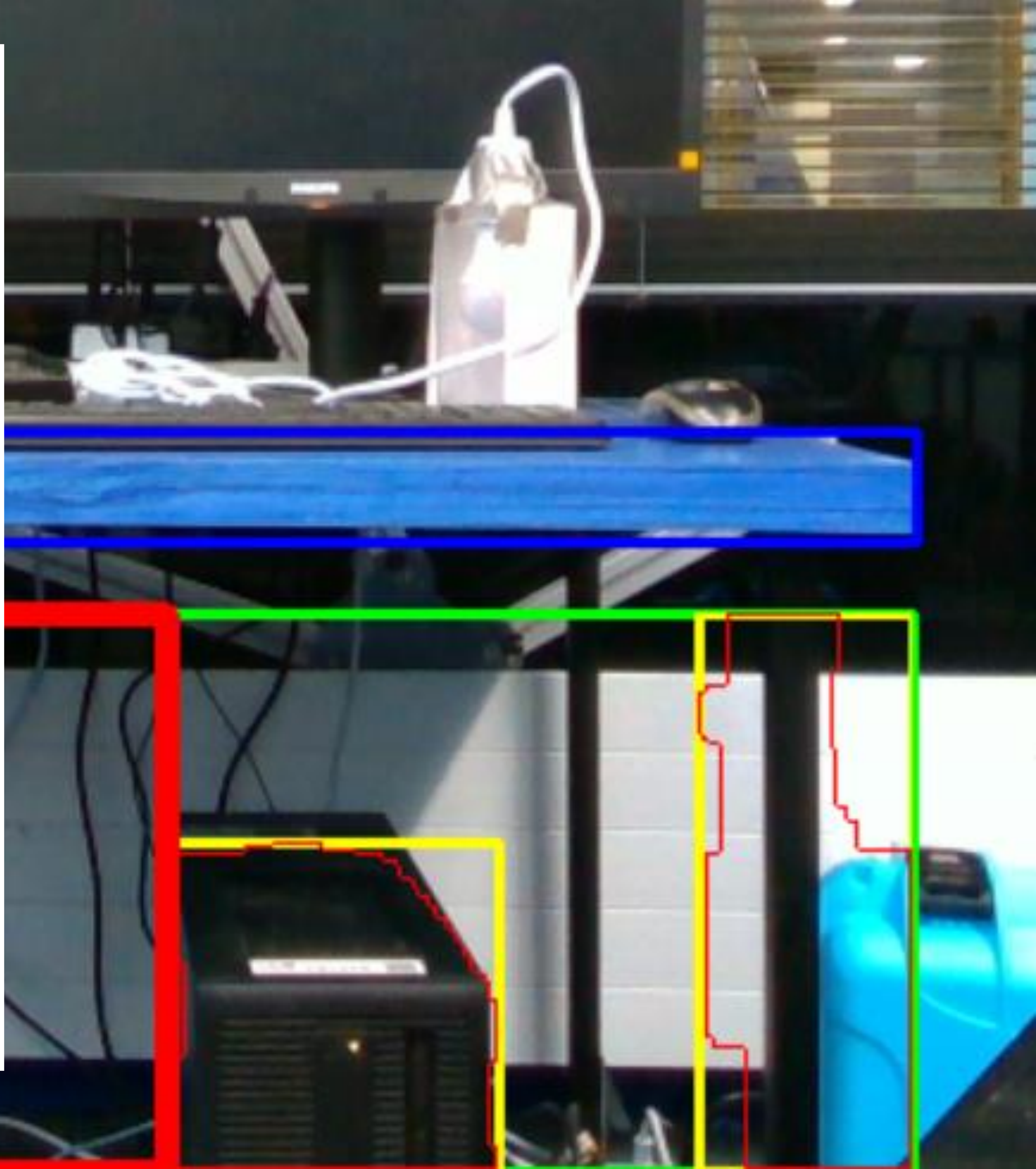
- S. Jain and B. Argall [1]
  - Kinect camera
    - Has to mounted high above chair
  - Point cloud method
  - Good detection of a rectangular and circular desks
  - Cannot detect obstacles hidden beneath desk



[1]
S. Jain and B. Argall, ‘Automated perception of safe docking locations with alignment information for assistive wheelchairs’, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 4997–5002.

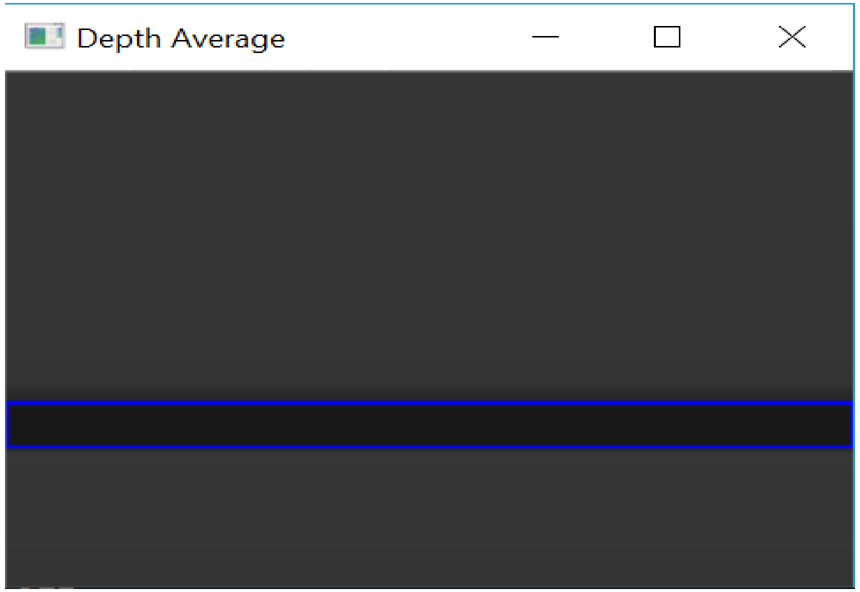
Outline

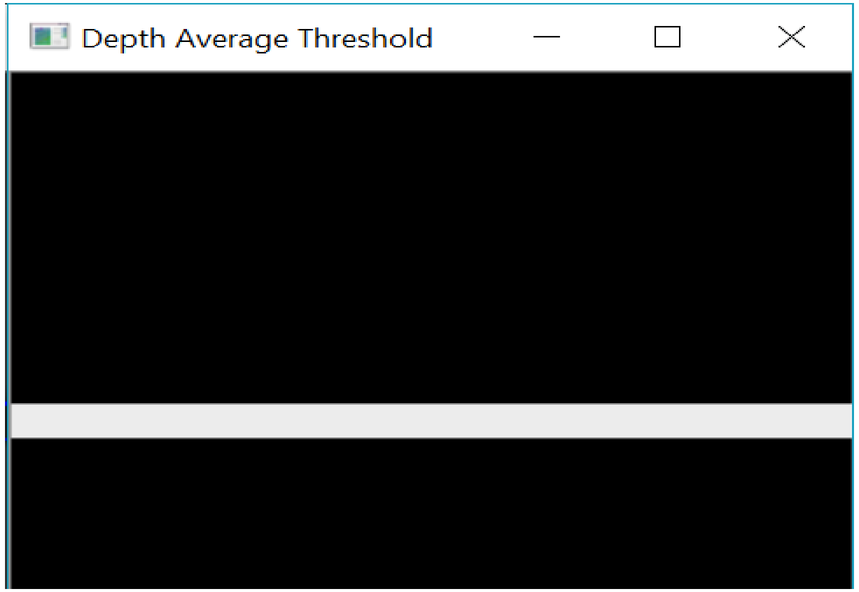
- **D435 Depth Camera**
  - Colour and Depth frames aligned
- **Finding the height of the front of the desk**
  - Horizontal convolution
  - Blur
  - Depth thresholding
  - Contours
- **Finding the width of the desk**
  - Depth thresholding
  - Morphological operations
  - Contours
- **Calculating the distance and angle of wheelchair relative to desk**
  - Camera intrinsics
  - Pixel to point deprojection
- **Detect any obstacles under the desk**
  - Depth thresholding
  - Morphological operations
  - Contours
- **Find the most suitable docking location**
  - Column scanning
- **Plot images**



Height of Desk

- Horizontal convolution
  - Average each row of depth values
- Blur
  - Kernel size (1,5)
- Depth thresholding
  - Threshold limits are the closest averaged depth +/- 20%
- Contours
  - Largest contour is assumed to be the desk



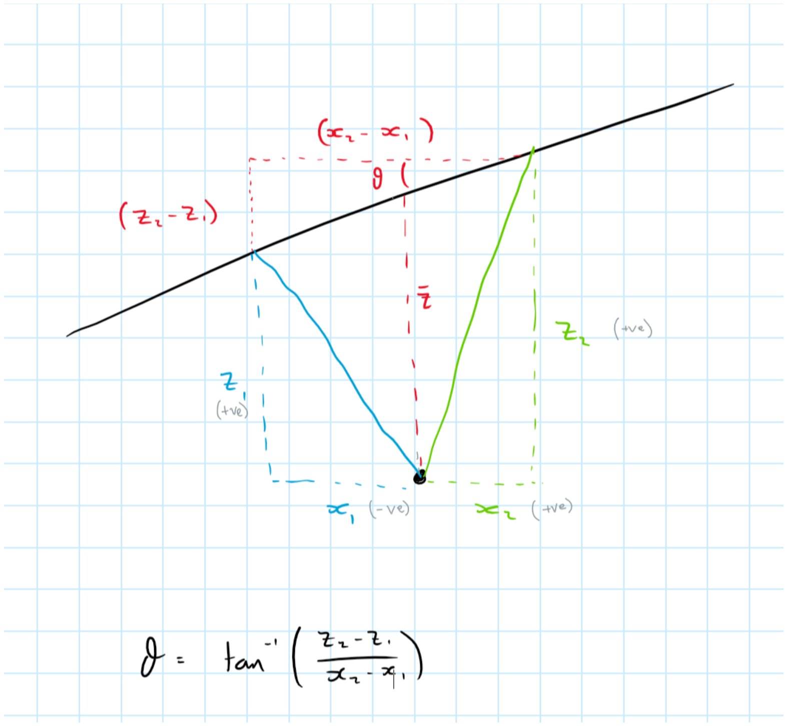


Width of Desk

- Depth thresholding depth image over region of interest (ROI)
- Morphological operations
- Contours
  - Largest contour assumed to be front of the desk.

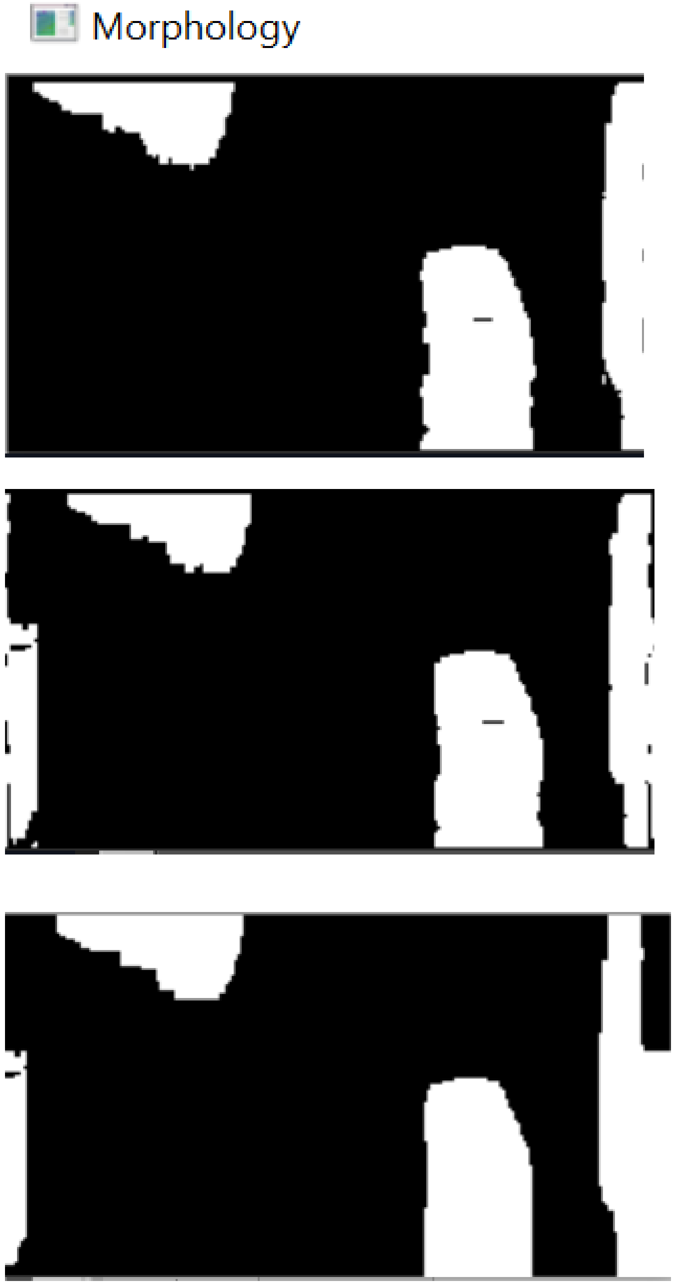
Distance/Angle of Desk

- Camera intrinsics
- Pixel to point deprojection
  - Takes a pixel and returns the 3D world coordinates.
  - 2 points are used
  - Trigonometry to calculate angle



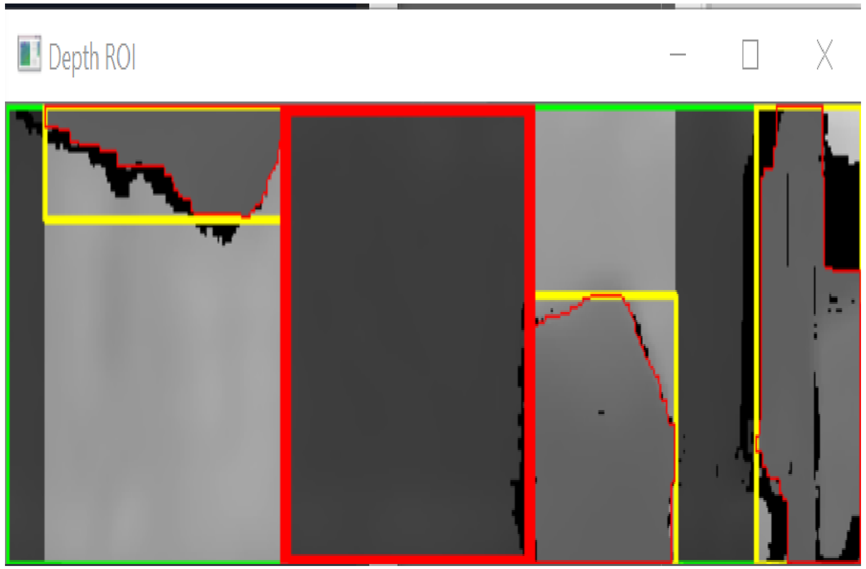
Obstacle Detection

- ROI under desk identified
  - Depth thresholding
  - Morphological operations
    - Opening 2 iterations
    - Closing 10 iterations
  - Bounding rectangle of contours used to highlight obstacles



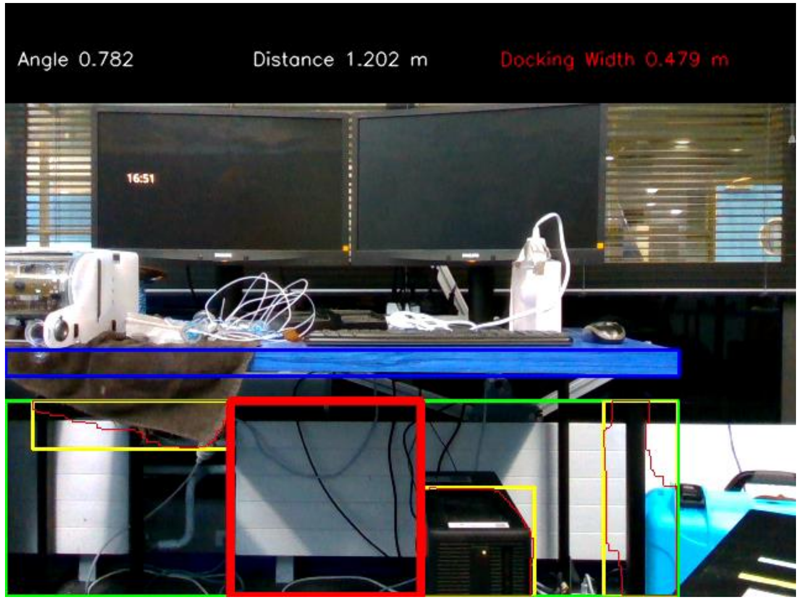
Docking Location

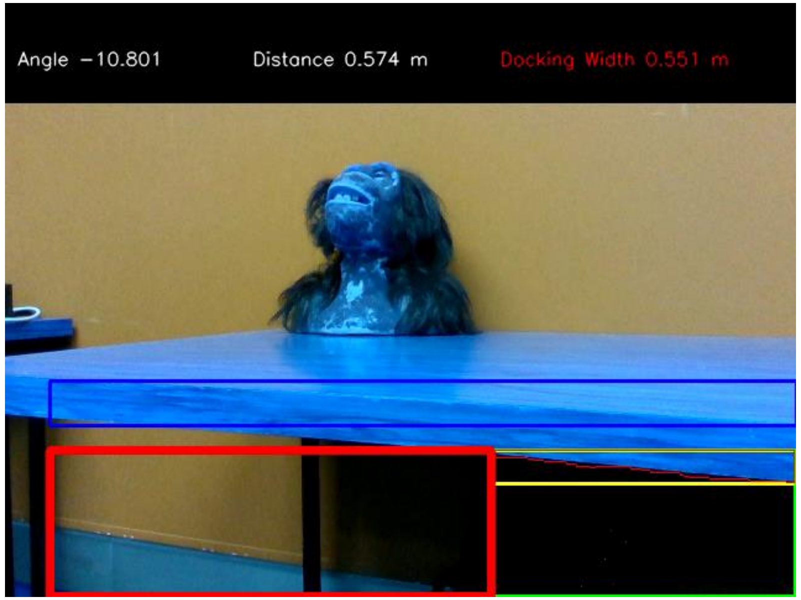
- Each column of the closed image is scanned to identify if there are no white pixels
- Adjacent columns that meet this criteria are grouped
- Largest group assumed to be most suitable docking location
- Physical size calculated



Demo

- Desk (blue)
- Under Desk (green)
- Obstacles
  - Contour (thin red)
  - Bounding rectangle (yellow)
- Docking area
  - Too small (red)
  - Big enough (white)

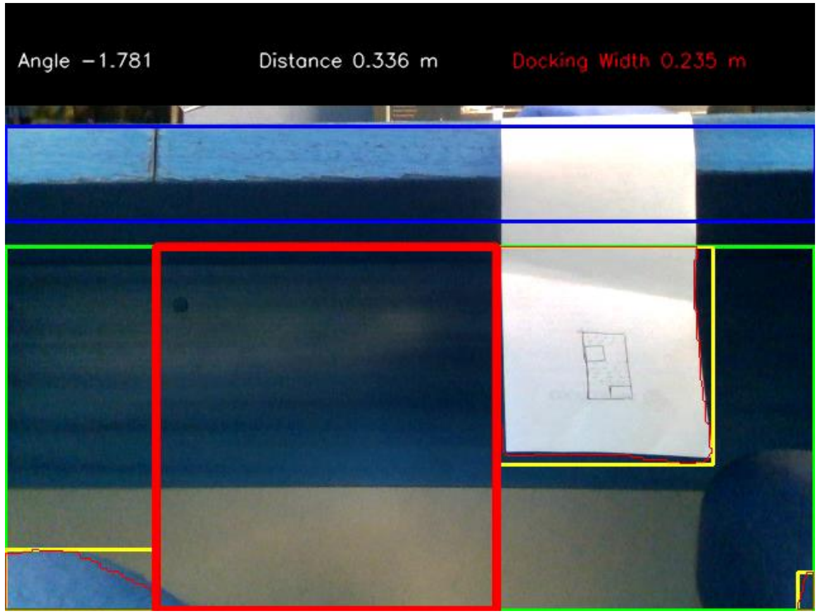


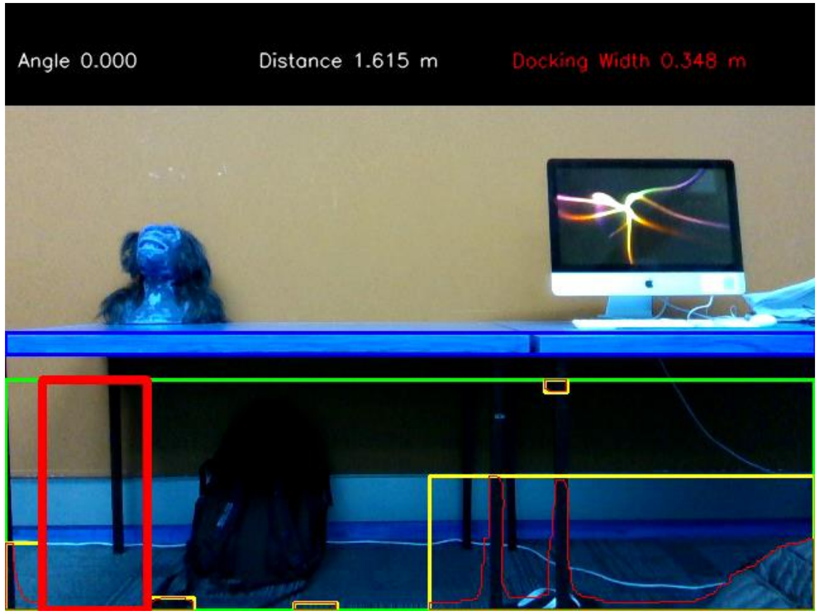


Results

- Angle accurate within 1 degree up to +/- 15^o from perpendicular
- Distance accurate within 0.01m between 0.11m - 10m[1]
- No limit on obstacles detected
- Desk can be partially or fully in view as long as it occupies >50% of the image horizontally

[1]
'Intel® RealSense™ Depth Camera D435 Product Specifications', *Intel® ARK (Product Specs)*. [Online]. Available: https://ark.intel.com/products/128255/Intel-RealSense-Depth-Camera-D435. [Accessed: 17-Apr-2018].





Algorithms

- Gaussian Blur
- Convolution
- Thresholding
- Morphological operations
- Contours
- Deprojection
- Etc.

```python
"""
Find all unobstructed points beneath desk
"""
clear_list = []
clear_index = []

alpha = 0.6
roi_overlay = roi_depth.copy()


for pixel_x in range(np.shape(closing)[1]):
    clear = True
    for pixel_y in range(np.shape(closing)[0]):
        if closing[pixel_y, pixel_x] != 0:
            clear = False
            break
    clear_list.append(clear)
    if clear == True:
        cv2.rectangle(roi_overlay,(pixel_x,0),(pixel_x,np.shape(closing)[0]),(0,0,0),-1)

# apply the overlay
cv2.addWeighted(roi_overlay, alpha, roi_depth, 1 - alpha,0, roi_depth)


"""
Find size of each docking location
"""
index1 = 0
index2 = 0
index_diff = 0
for i in range(1, len(clear_list)):

    if clear_list[i-1] == False and clear_list[i] == True:
        index_diff = 0
        index1 = i
    elif clear_list[i-1] == True and clear_list[i] == True:
        index_diff += 1
    elif clear_list[i-1] == True and clear_list[i] == False:
        index2 = i
        clear_index.append((index1,index2,index_diff))

    if (i == len(clear_list)-1 and clear_list[i] == True):
        index2 = i
        clear_index.append((index1,index2,index_diff))


"""
Find largest docking location
"""
docking_width = 0
for i in range(len(clear_index)):
    if clear_index[i][2] > docking_width:
        docking_width = clear_index[i][2]

        docking_location = (clear_index[i][0],clear_index[i][1])
```