CS 378: Autonomous Intelligent Robotics (FRI)

Dr. Todd Hester

Are there any questions?

Logistics

- Summer FRI Fellowships
  - $2500 stipend
  - Deadline is Monday
- Fall Course
- No Class on Tuesday
- Progress Reports Due Next Thursday

Progress Reports

- Due Next Thursday
- 4 pages, double spaced
- Academic paper format
  - Title, abstract, section headings, possibly citations
- Largely a revision of your proposal
  - Still what you plan to do, steps in plan, team members, etc
  - Can re-use much of the text
- Update/revise your plan and goals
- Section on previous work. What have you completed?
- Show some preliminary result

Today

- Vision
- Group Updates

Viewing images

- Start *the segbot*
- Start the default image viewer - provided through the ROS image pipeline
  - *rosrun image_view image_view image:=/kinect/rgb/image_color*
- Play back a bag file
  - *rosbag play -l <bag_file>*

OpenCV

- OpenCV is an open source computer vision library
- A large part of vision research is typically done in Matlab. As a result, there is a lot more code available through Matlab than OpenCV.
- However, we like OpenCV because it is *easier* to integrate vision into robotics through it instead of Matlab-based approaches.
  - Additionally, a lot of OpenCV code has been written fairly efficiently
-

OpenCV

- http://opencv.org

- **OpenCV** (**Open** Source **C**omputer **V**ision) is an open source implementation of a number of popular computer vision algorithms
  - Face Detection
  - Pedestrian Detection
  - Local Feature Extraction
  - ...

OpenCV

- OpenCV has a lot of implementations of some really cool algorithms.
- Also some basic filters and image manipulation
  - Threshold
  - Image Derivatives (Sobel) and Edge Detection (Canny)
  - Dilate and Erode
  - Flood Fill
  -

cv_bridge

- http://www.ros.org/doc/api/cv_bridge/html/c++/index.html
- Converts between ROS Image message formats and OpenCV image formats
- ROS -> OpenCV
  - *toCvCopy()*
  - *toCvShare()*
- OpenCV -> ROS
  - *toImageMsg()*

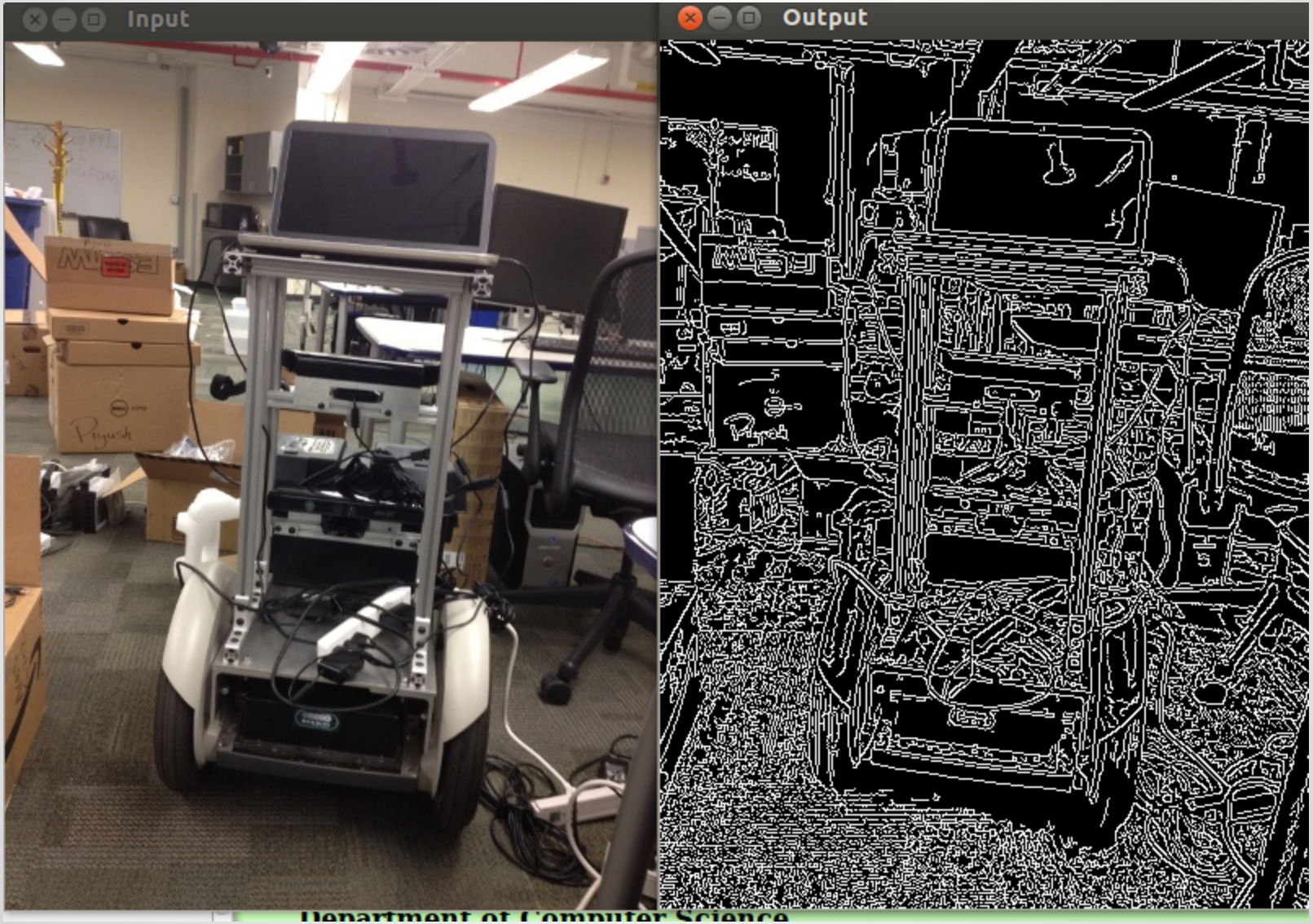
```
int main(int argc, char *argv[]) {
  ros::init(argc, argv, NODE);

  cv::namedWindow("Input");
  cvResizeWindow("Input", 320, 240);

  cv::namedWindow("Output");
  cvResizeWindow("Output", 320, 240);

  cvStartWindowThread();

  ros::NodeHandle node;
  image_transport::ImageTransport it(node);

  std::string image_topic = node.resolveName("usb_cam/image_raw");
  image_transport::Subscriber center_camera =    it.subscribe(image_topic, qDepth_, &processImage);

  ROS_INFO(NODE ": starting main loop");

  ros::spin();                          // handle incoming data

  ROS_INFO(NODE ": exiting main loop");

  return 0;
}
```

```
void processImage(const sensor_msgs::ImageConstPtr &msg) {
  // Get a reference to the image from the image message pointer
  cv_bridge::CvImageConstPtr imageMsgPtr = cv_bridge::toCvShare(msg, "bgr8");
  cv::Mat outputImage;
  callFilter(imageMsgPtr->image, outputImage);
  cv::imshow("Output", outputImage);
  cv::imshow("Input", imageMsgPtr->image);
}

void callFilter(const cv::Mat& inputImage, cv::Mat& outputImage) {
  switch (method_) {
        case CANNY:
        getCannyImage(inputImage, outputImage);
        break;
        case FLOOD:
        getFloodFillImage(inputImage, outputImage);
        break;
        case THRESHOLD:
        getThresholdImage(inputImage, outputImage);
        break;
  }
}
```

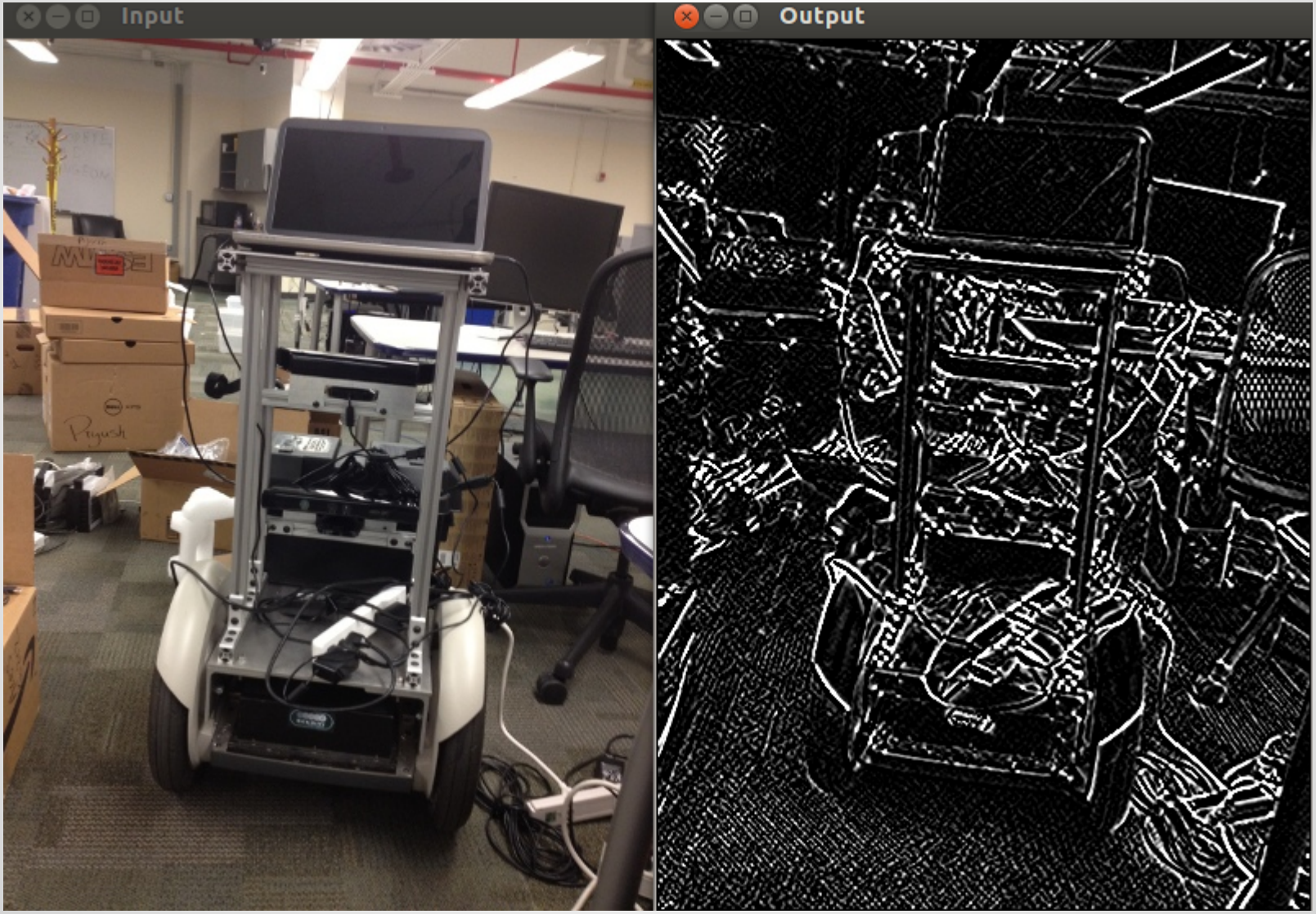
```
void getCannyImage(const cv::Mat& inputImage, cv::Mat& outputImage) {
  // Get a gray image - quite a bit of vision processing is done on grayscale images
  cv::Mat grayImage;
  cv::cvtColor(inputImage, grayImage, CV_RGB2GRAY);

  // Get an edge image - here we use the canny edge detection algorithm to get the edges
  double threshold1 = 20;
  double threshold2 = 50;
  int apertureSize = 3;

  // The smallest of threshold1 and threshold2 is used for edge linking,
  // the largest - to find initial segments of strong edges.  Play around
  // with these numbers to get desired result, and/or pre-process the
  // image, e.g. clean up, sharpen, blur).
  cv::Canny(grayImage, outputImage, threshold1, threshold2, apertureSize);
}
```

```
/* http://opencv.jp/opencv-2.2_org/cpp/imgproc_miscellaneous_image_transformations.html#cv-threshold */
void getThresholdImage(const cv::Mat& inputImage, cv::Mat& outputImage) {
  // Get a gray image - quite a bit of vision processing is done on grayscale images
  cv::Mat grayImage;
  cv::cvtColor(inputImage, grayImage, CV_RGB2GRAY);

  int thresh = 128;
  double maxVal = 255;
  int thresholdType = CV_THRESH_BINARY;

  cv::threshold(grayImage, outputImage, thresh, maxVal, thresholdType);
}
```

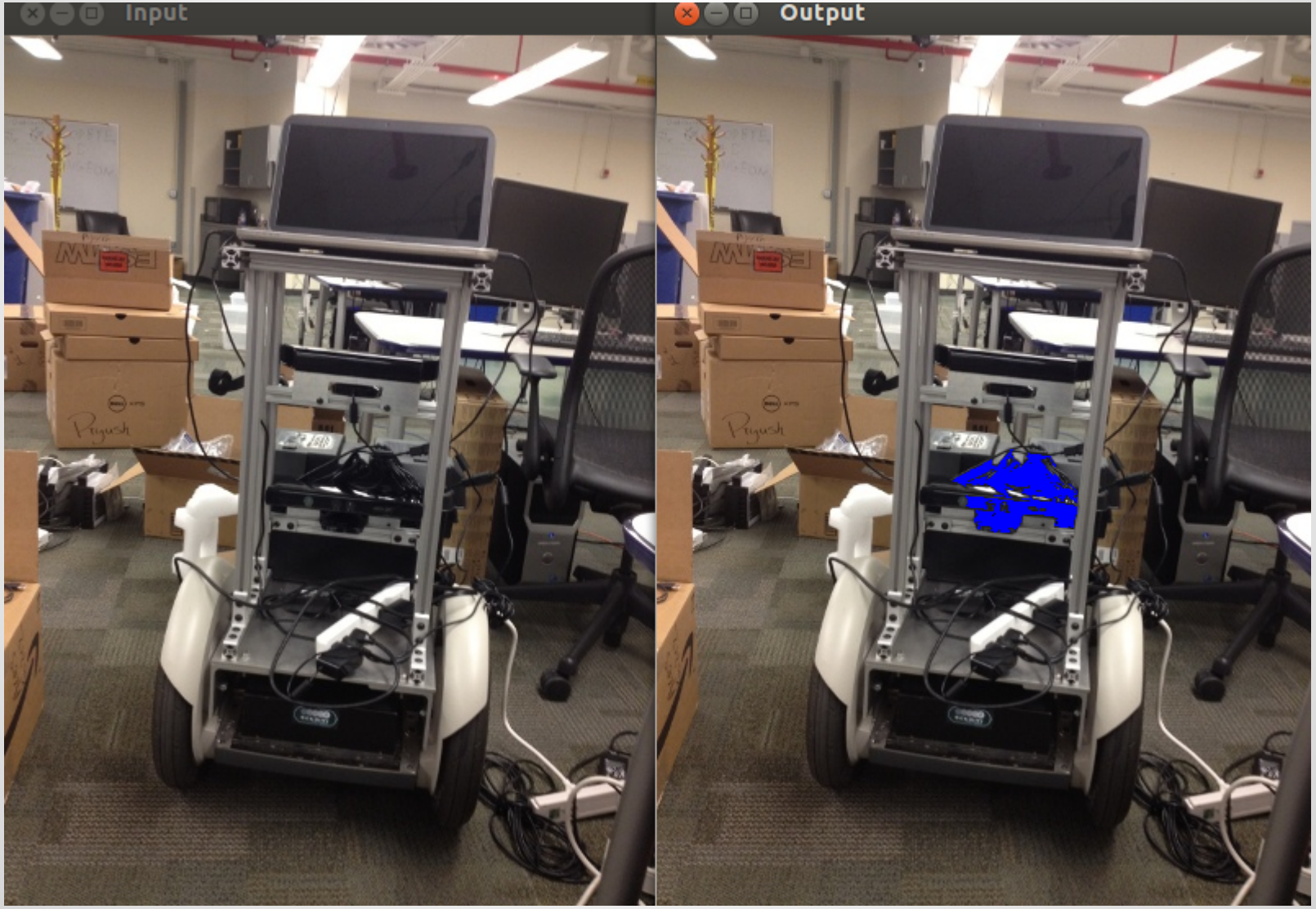
Canny Edge Detection



Sobel



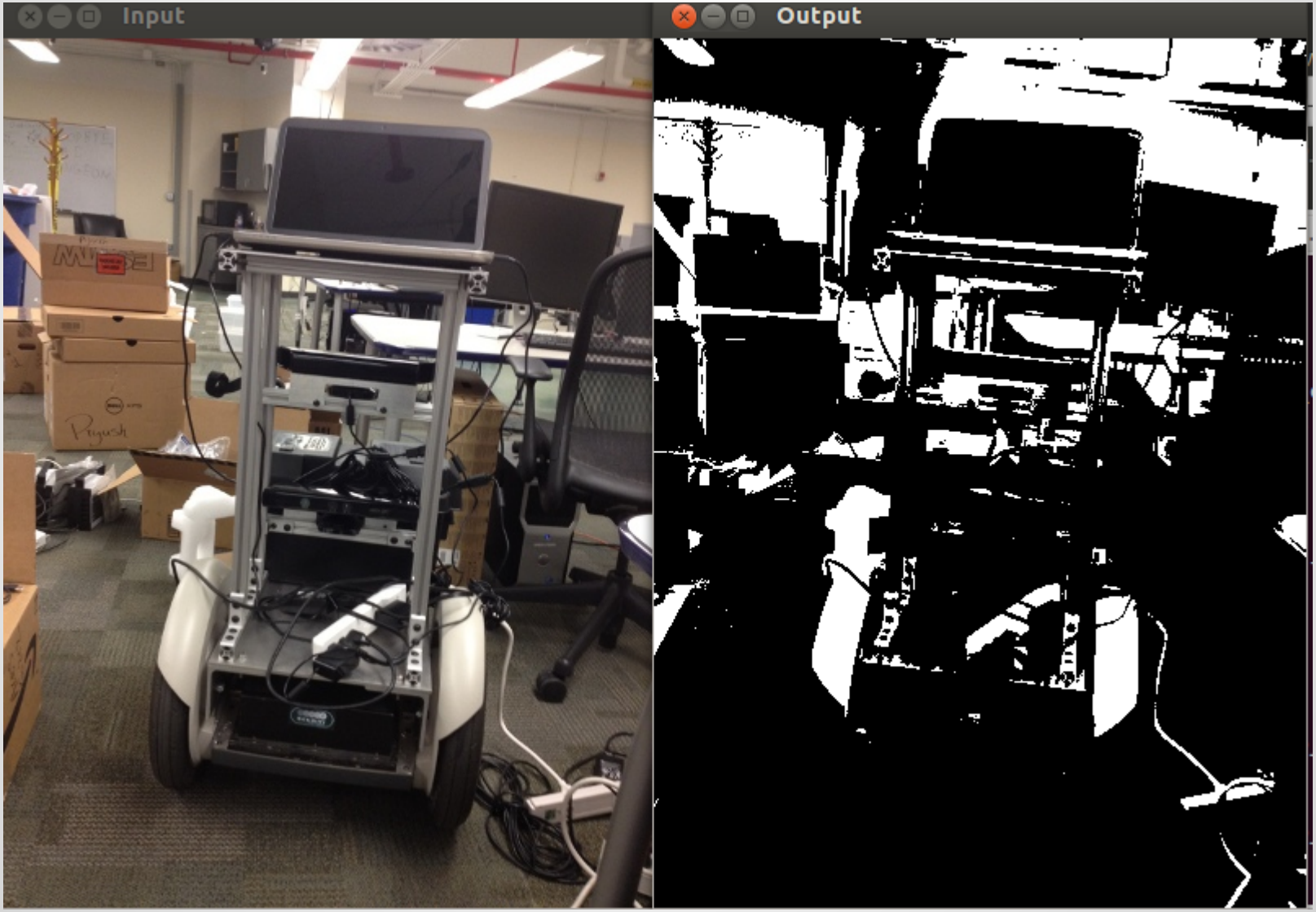
Flood Fill



Erode

Dilate

Threshold



Group Updates