ERDC/GRL TR-21-6

US Army Corps
of Engineers®
Engineer Research and
Development Center

ROS Integrated Object Detection for SLAM in Unknown, Low-Visibility Environments

Benjamin Christie, Osama Ennasr, Garry Glaspell

November 2021

Geospatial Research Laboratory

The U.S. Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdc.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at https://erdclibrary.on.worldcat.org/discovery.

ERDC/GRL TR-21-6
November 2021

ROS Integrated Object Detection for SLAM in Unknown, Low-Visibility Environments

Benjamin Christie, Osama Ennasr, Garry Glaspell

U.S. Army Engineer Research and Development Center (ERDC)
Geospatial Research Laboratory (GRL)
7701 Telegraph Road
Alexandria, VA 22315-3864

Final Report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under Program Element Number 622146
Project Number AT9 Task Number 01

Abstract

Integrating thermal (or infrared) imagery on a robotics platform allows Unmanned Ground Vehicles (UGV) to function in low-visibility environments, such as pure darkness or low-density smoke. To maximize the effectiveness of this approach we discuss the modifications required to integrate our low-visibility object detection model on a Robot Operating System (ROS). Furthermore, we introduce a method for reporting detected objects while performing Simultaneous Localization and Mapping (SLAM) by generating bounding boxes and their respective transforms in visually challenging environments.

Contents

Abstract ii

Figures iv

Preface v

Acronyms and Abbreviations vi

1 Introduction 1

1.1 Background 1
1.2 Objective 1
1.3 Approach 2

2 ROS Integration 6

3 Conclusion and Future Implementations 11

4 Installation 12

4.1 Configuration 14
4.2 Launch 15

5 Code Listings 16

References 30

Report Documentation Page 31

Figures

Figures

1 Training results per epoch for YOLOv3-SPP 3
2 Training results per epoch for YOLOv3-tiny 4
3 Final performance results for the YOLOv3-SPP configuration after 300 epochs 4
4 Final performance results for the YOLOv3-tiny configuration after 300 epochs 5
5 YOLOv3 algorithm falsely identifying a dog as a car due to the insufficient training data for the Dog object class compared to Car 6
6 Comparison of unclassified thermal image (*left*) and the output of a well-trained YOLOv3 model (*right*). The object-detection algorithm is able to correctly classify and localize objects in a complex environment that can be challenging even to a human eye 7
7 YOLOv3 detecting objects alongside cartographer_ros and subt_solution_launch in a simulated environment 8
8 Generating bounding box around detected object in the map along with appropriate transforms in a simulation environment 9
9 FLIR camera integrated into RTABMap 10
10 Successful identification of car (left) and misclassification based on atmospheric conditions and orientation (*right*) 11

Preface

This study was conducted for the Engineer Research and Development Center, Geospatial Research Laboratory (ERDC-GRL) of the U.S. Army Corps of Engineers under Project 622146/AT9/01, "Tactical Geospatial Information Capabilities (TGIC)," Task "Tactical Data Generation and Processing." The technical monitor was Dr. Jean Nelson.

The work was performed by the Data Representation Branch of the Topography Imagery and Geospatial Research Division, ERDC-GRL. At the time of publication, Mr. Vineet Gupta was Branch Chief; Ms. Jennifer Smith was Division Chief; and Mr. Ritchie Rodebaugh was the Technical Director for Geospatial Research and Engineering (GRE) business area. The Deputy Director of ERDC-GRL was Ms. Valerie Carney and the Director was Mr. Gary Blohm.

COL Teresa A. Schlosser was Commander of ERDC, and Dr. David W. Pittman was the Director.

Acronyms and Abbreviations

Acronym	Term
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
DARPA	Defense Advanced Research Projects Agency
FLIR	Forward Looking InfraRed
FPS	Frames-Per-Second
GB	Gigabyte
GPU	Graphics Processing Unit
IOU	Intersection over Union
lidar	Light Detection and Ranging
mAP	Mean Average Precision
MSCOCO	Microsoft Common Objects in Context
R-CNN	Regions with CNN
RGB	Red-Green-Blue
ROS	Robot Operating System
RTABMap	Real-Time Appearance-Based Mapping
SLAM	Simultaneous Localization and Mapping
SPP	Spatial Pyramid Pooling
SWaP	Size, Weight, and Power
YOLO	You Only Look Once

1 Introduction

1.1 Background

Object detection is a growing field in computer science, and approaches to object detection have greatly improved over the past 20 years. Deep learning approaches have been particularly successful in classifying and localizing different objects in an image. Of those deep learning approaches, the Convolutional Neural Network (CNN) algorithm is effective at differentiating objects. One of the first large and successful applications of CNNs to object detection is the Regions with CNNs (R-CNN) proposed in (Girshick et al. 2014). However, this approach is relatively slow, requiring a CNN-based feature extraction pass on each of the candidate regions generated by the region proposal algorithm. To address this issue, researchers proposed extensions to speed up the process, resulting in Fast R-CNN (Girshick 2015) and Faster R-CNN (Ren et al. 2017). In particular, Faster R-CNN achieves near state-of-the-art results in object recognition tasks at a rate of five frames-per-second (FPS) (Ren et al. 2017), which is significantly faster than R-CNN.

For autonomous robotic applications, however, these speeds are still too slow, and alternative real-time (five or more FPS) algorithms with similarly high accuracy are required. The "You Only Look Once" (YOLO) object detection algorithm was developed to be a faster alternative (Redmon et al. 2016). Compared to R-CNN, the YOLO algorithm is much faster, achieving object detection in real-time. The accuracy of the YOLO algorithm is also remarkably high, making it a very attractive option for implementation in real-world settings. The YOLO algorithm was then improved upon to allow it to predict 9000 object classes in YOLOv2 (Redmon and Farhadi 2017), and additional incremental improvements followed in YOLOv3 (Redmon and Farhadi 2018), YOLOv4 (Bochkovskiy, Wang, and Liao 2020), and YOLOv5. In short, YOLO is faster because it takes in the whole image at once, in contrast to other approaches that leverage regions to localize the object within the image.

1.2 Objective

Typically, Red-Green-Blue (RGB) images are used for object detection algorithms such as YOLO. However, these algorithms are not limited to

RGB. In fact, relying exclusively on RGB images can cause these algorithms to become unreliable when operating in low-visibility conditions (e.g., night-time, insufficient lighting, or smoke). Since these conditions are commonly encountered in subterranean environments such as tunnels and caves, thermal or infrared imagery is necessary (apart from Light Detection and Ranging [lidar] systems) for appearance-based mapping to be conducted. To that end, this report presents a novel implementation of the YOLO algorithm designed for low-visibility environments on a custom low size, weight, and power (SWaP) payload.

1.3 Approach

For an autonomous operation, real-time algorithms with a high accuracy are optimal. Solutions such as Faster R-CNN, YOLOv2, YOLOv3, and YOLOv5 were considered. YOLOv3 was ultimately decided upon since it is lightweight, allows for real-time operation, offers the most community support, and is easily configured. Furthermore, YOLOv3 can be integrated with a Robot Operating System (ROS) and leverage newer versions of Compute Unified Device Architecture (CUDA) (a set of graphics processing unit (GPU)-based tools for training and optimizing artificial intelligence [AI] models) by incorporating and configuring the ROS package `darknet_ros`, not just CUDA 8.

For thermal or infrared imagery, the Forward Looking InfraRed (FLIR) Boson 320 thermal camera was selected as it fits the low-SWaP requirements for our platform. This camera was then used to test YOLOv3 in low-visibility environments. To train the AI model, we utilized the free "FLIR Thermal Dataset for Algorithm Training" provided by the manufacturer (Teledyne FLIR, n.d.). The dataset contains 25 gigabyte (GB) of synced annotated thermal images and videos (along with nonannotated RGB images for reference), and contains annotations in Microsoft Common Objects in Context (MSCOCO) format for the object classes: Person, Car, Bicycle, Dog, and Other Vehicle. The available frames are split between day (60%) and night (40%) driving on Santa Barbara, California, area streets and highways during November to May with clear to overcast weather (Teledyne FLIR, n.d.). The FLIR dataset achieved a mean average precision (mAP) of 0.794, 0.580, and 0.856 for people, bicycles, and cars, respectively.

First, we evaluated two configurations of YOLOv3: YOLOv3-tiny, which is generally a faster but less accurate configuration, and YOLOv3-SPP, which

utilizes Spatial Pyramid Pooling (SPP) to offer higher accuracy, but lower speeds. For our testing, we used only 8-bit thermal images contained in the dataset. Whether an image was used for validation or for training was randomized but consistent so that the comparison between the YOLOv3-tiny and YOLOv3-SPP configurations was consistent. After randomization, a total of 9206 images were used for training the model, and 1022 images used for validation.

The training process for the configurations tested were comparable, although YOLOv3-SPP had higher precision and recall values in much fewer epochs. However, the YOLOv3-tiny configuration training process was much faster, taking about a fifth of the time per epoch compared to YOLOv3-SPP. These training results are detailed in Figure 1 and Figure 2. Note that more epochs were allotted for the YOLOv3-SPP since the mAP value was highly variable near the 300th epoch.

Figure 1. Training results per epoch for YOLOv3-SPP.

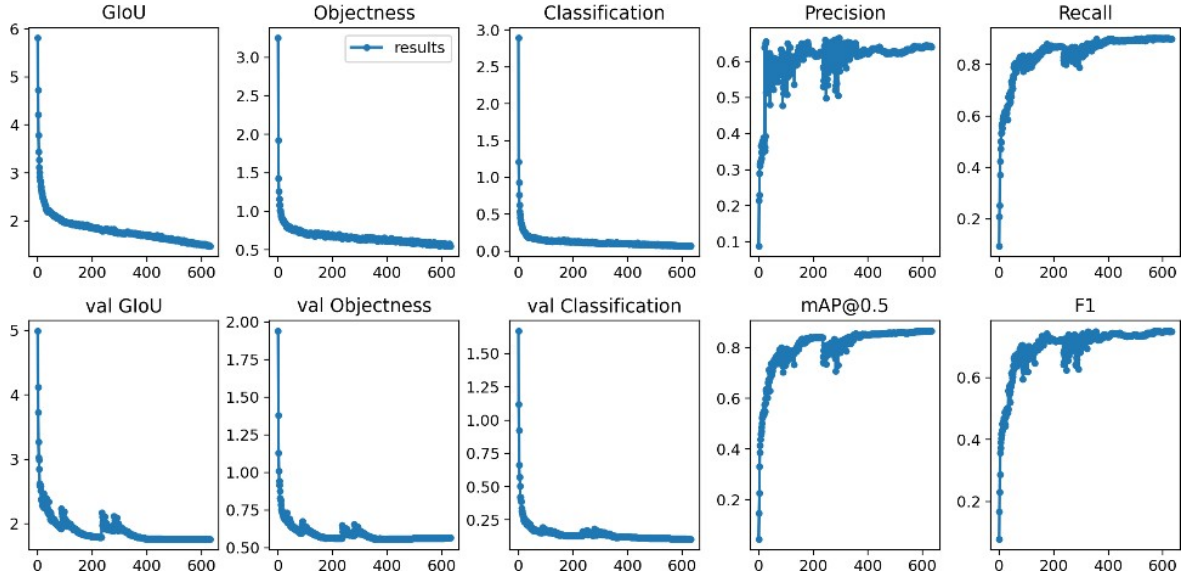
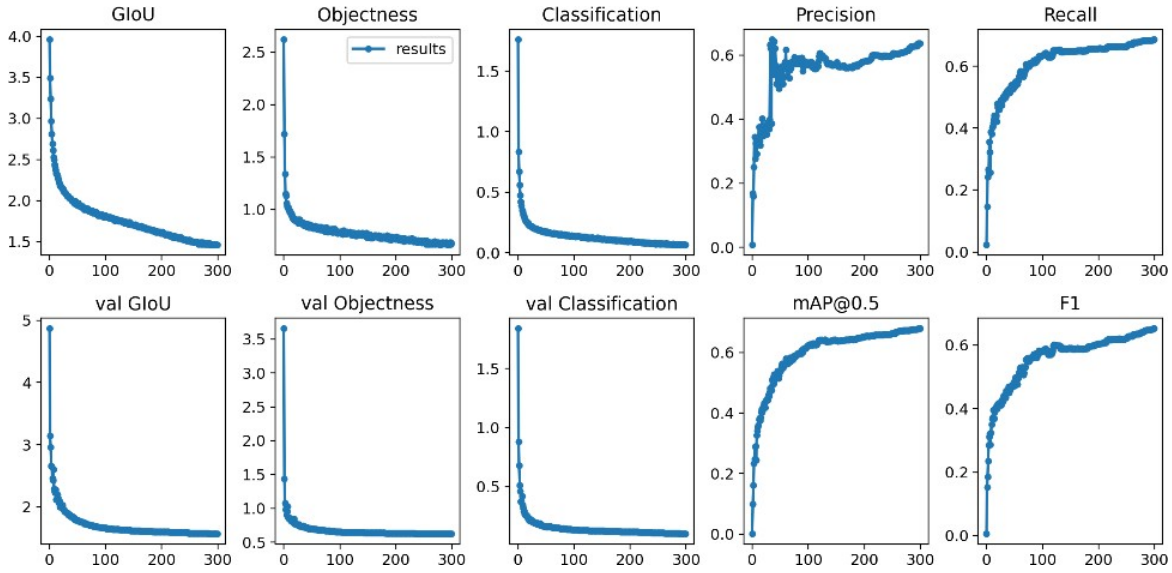


Figure 2. Training results per epoch for YOLOv3-tiny.



To compare object-detection performance, both configurations were trained from an empty set of weights on all images. After 300 epochs of training on a GeForce RTX2060/PCiE/SSE2 GPU and Intel Core i7-9750H cpu at 2.60 GHz × 12, the YOLOv3-tiny configuration had reached a mAP at a 0.5 Intersection over Union (IoU) of 0.582, a precision of 0.611, and a recall of 0.592.[1] And after 300 epochs, the YOLOv3-SPP configuration reached a mAP at .5 IoU of 0.805, a precision of 0.600, and a recall of 0.815. Figure 3 and Figure 4 depict screenshots of the final performances of the YOLOv3-SPP and YOLOv3-tiny configurations, respectively. It should be noted that the Bicycle and Dog classes have very different results when compared to the other classes. This is likely due to the fact that there are less images that contain these object classes within the provided FLIR dataset.

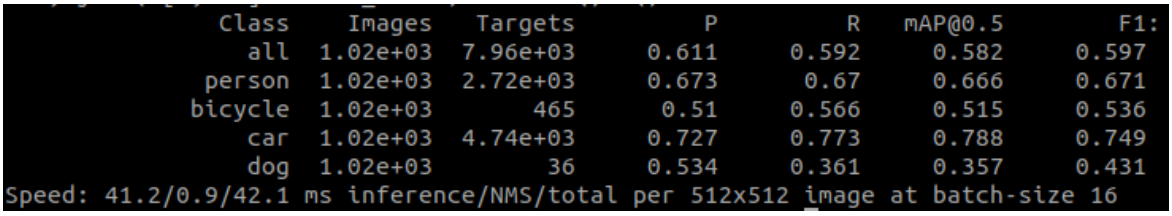
Figure 3. Final performance results for the YOLOv3-SPP configuration after 300 epochs.

Class	Images	Targets	P	R	mAP@0.5	F1:
all	1.02e+03	7.96e+03	0.722	0.872	0.845	0.789
person	1.02e+03	2.72e+03	0.716	0.9	0.884	0.797
bicycle	1.02e+03	465	0.632	0.826	0.778	0.716
car	1.02e+03	4.74e+03	0.766	0.931	0.925	0.84
dog	1.02e+03	36	0.774	0.833	0.791	0.802

Speed: 321.7/1.3/323.0 ms inference/NMS/total per 512x512 image at batch-size 16

[1] IoU stands for Intersection over Union, a measure of "correctness" of the resulting bounding box in object detection problems. It is generally agreed that an IoU of 0.5 is the cutoff between true positives and false positives.

Figure 4. Final performance results for the YOLOv3-tiny configuration after 300 epochs.



Class	Images	Targets	P	R	mAP@0.5	F1:
all	1.02e+03	7.96e+03	0.611	0.592	0.582	0.597
person	1.02e+03	2.72e+03	0.673	0.67	0.666	0.671
bicycle	1.02e+03	465	0.51	0.566	0.515	0.536
car	1.02e+03	4.74e+03	0.727	0.773	0.788	0.749
dog	1.02e+03	36	0.534	0.361	0.357	0.431

Speed: 41.2/0.9/42.1 ms inference/NMS/total per 512x512 image at batch-size 16

While the YOLOv3-SPP configuration performs much better than the YOLOv3-tiny configuration in mAP and recall metrics, it should be noted that the YOLOv3-tiny configuration ran at 220 FPS, whereas the YOLOv3-SPP configuration only ran at 20 FPS on the same hardware. For a real-time autonomous robotic application with low SWaP requirements, 20 FPS is rather slow, and the YOLOv3-tiny could be more advantageous. However, if hardware improves, then YOLOv3-SPP configurations can be retested in the field. Additionally, it should be noted that the cameras used in this configuration ran at approximately 30 FPS, so the YOLOv3-tiny configuration analyzed multiple copies of the same frames. The intent was to match the object recognition rate to the rate of the navigation stack. This would allow us to efficiently leverage object recognition to change the robot's behavior if an object of interest was identified. Presently, YOLOv3-tiny meets this requirement.

These results suggest that YOLOv3 can adequately detect objects for robotic use when trained on enough data. The quality and amount of data provided is crucial, however. For instance, if the provided data set is imbalanced, and the neural network is trained on several thousand images of cars but only a few hundred images of dogs, then images of dogs can frequently be marked incorrectly as cars (Figure 5). This can be very dangerous when applied to the warfighter, as labeling an object could be mission critical. However, when trained on sufficient data, these inaccuracies are much less frequent, and the algorithm’s performance is much better at spotting objects that are challenging for a human eye to distinguish (Figure 6). To fully leverage the benefits of fast object detection, we attempted to localize the recognized objects into a 3D space that the robot can understand. As a result, this capability allows the robot to set way points near the object of interest. Specifically, this could be used to follow a dynamic object as it moves.

2 ROS Integration

Integrating YOLOv3 into ROS is crucial for object detection to be fully utilized in robotics, as it allows for integration with other ROS-based capabilities that lead to exciting new ideas such as integration with Real-Time Appearance-Based Mapping (RTABMap)_ROS for Simultaneous Localization and Mapping (SLAM applications. Installation instructions for YOLOv3 and integrating it with ROS and RTABMap are provided in Section 4.

We also considered the open-source ROS package `darknet_ros` provided by *leggedrobotics*. However, it does not natively support most versions of CUDA. To remedy this, we modified compute codes in the `CMakesList.txt` file and set the `LD_LIBRARY_PATH` and `PATH` environment variables in the Bash shell script. The details of this are contained in Section 4.

Figure 5. YOLOv3 algorithm falsely identifying a dog as a car due to the insufficient training data for the Dog object class compared to Car.

Deep-learning algorithms are highly dependent on the available computational resources. While training may be accomplished on more powerful hardware, issues could appear when implementing these algorithms on low-SWaP platforms. Specifically, if many processes with high memory usage are running simultaneously, `darknet_ros` may not perform optimally. Furthermore, `darknet_ros` may fail to launch larger configurations of

YOLOv3 (i.e., YOLOv3-SPP) due to insufficient memory. Thus, it may be advantageous, and sometimes critical, to train the models using the YOLOv3-tiny configuration, as well as utilize this lighter configuration in the field. However, on more powerful systems with less restrictions on power and computational resources, the default and SPP configurations will likely prove to be more advantageous for their better overall accuracy.

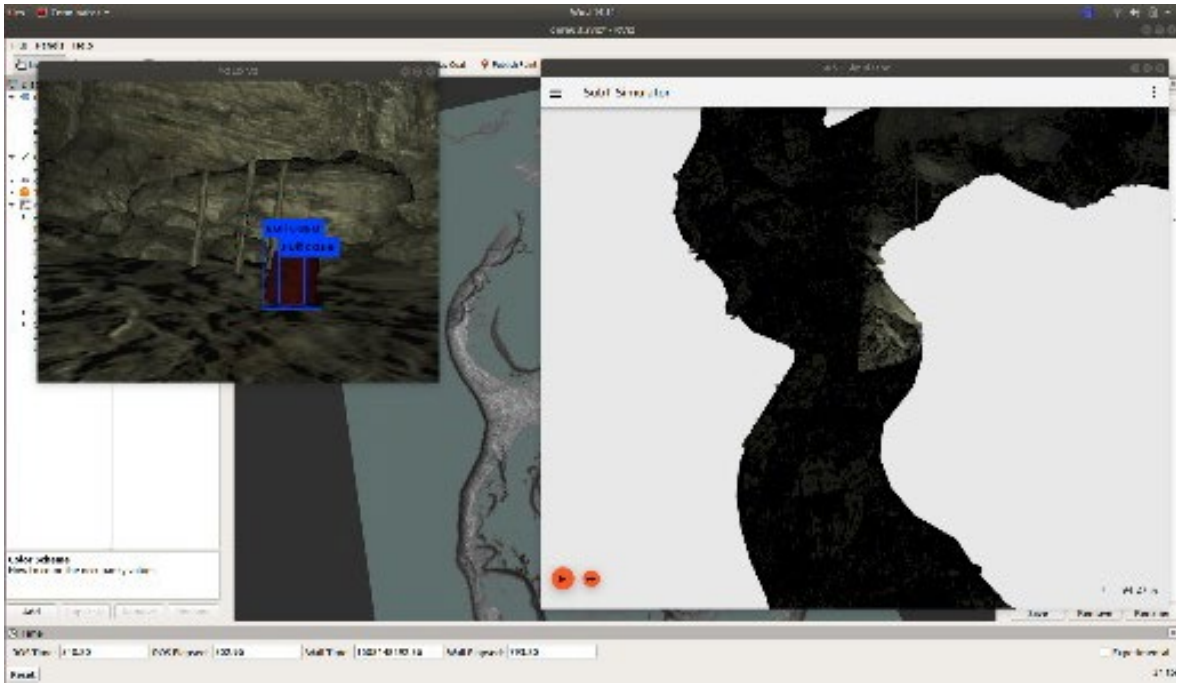
Figure 6. Comparison of unclassified thermal image (*left*) and the output of a well-trained YOLOv3 model (*right*). The object-detection algorithm is able to correctly classify and localize objects in a complex environment that can be challenging even to a human eye.

SLAM algorithms, such as RTABmap (Labbé and Michaud 2019) allow for a real-time construction and visualization of an unknown map, and can be combined with autonomous exploration approaches to improve situational awareness of unknown environments (Glaspell et al. 2020). For example, it is possible to further improve awareness by reporting detected objects to the SLAM algorithm and overlaying them on a map. Detected objects can be reported and marked by attaching a transform from each detected object to the **`/map`** frame generated by the SLAM algorithm. The **`subt_solution_launch/artifact_origin_reporter`** node provided by `subt` can be utilized for this as a proof of concept. The **`subt_solution_launch`** package is a demonstration of a potential solution to the Defense Advanced Research Projects Agency (DARPA) subterranean challenge, and many of the nodes provided in the package are useful for object recognition in mapping in general environments. Specifically, the **`artifact_origin_reporter`** node can be used to crop the point cloud within the bounding boxes produced by YOLOv3, and then calculate and report the origin of the point cloud to the user. A transform between the origin of the point cloud and the **`/map`** frame is then published. The **`subt_solution_launch`** was tested in

simulation and must be launched in the `subt_broker` in ignition. This tool published at approximately 2 Hz.

It should be noted that for this package to function properly, a point cloud, RGB image, and map topic must all be published. This can be done with mapping software such as Cartographer or RTABmap. Figure 7 demonstrates this process working in simulation with `cartographer_ros` mapping software. Note that the default YOLO3 pt weights was used for that demonstration instead of the custom FLIR model, although the end result was the same.

Figure 7. YOLOv3 detecting objects alongside cartographer_ros and subt_solution_launch in a simulated environment.



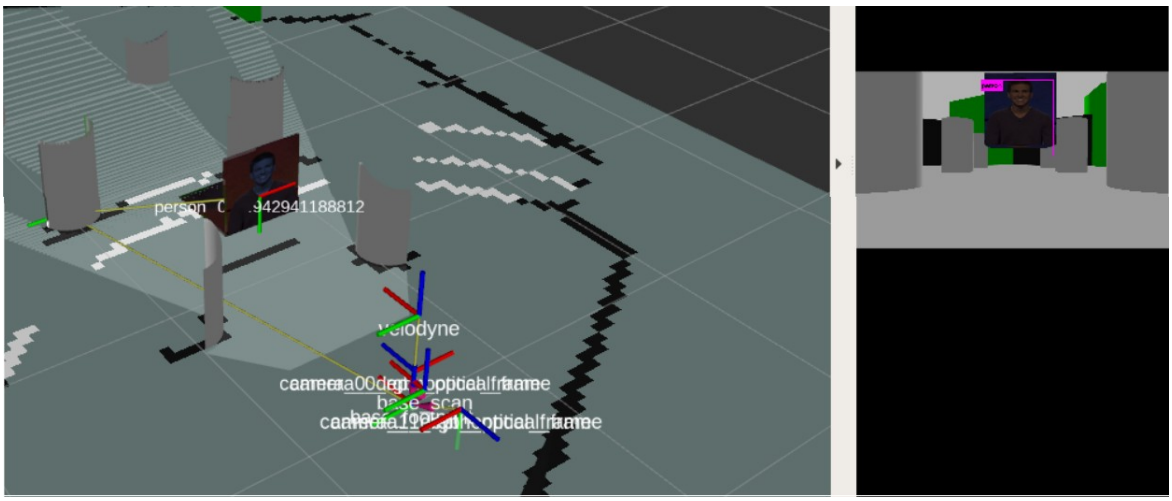
Once objects are detected by YOLOv3, a bounding box can be utilized to mark these detected objects on the generated map in real time. Layering bounding boxes on images can also be useful for different calculations in mapping and provides greater awareness of the environment. For example, by using the `RTABMap/rgbd_sync` nodelet to synchronize depth and RGB image topics, userscan synchronize the RGB image used for object detection with a depth image. This depth image can either be a raw depth image, or a point cloud topic converted to a depth image using the `RTABMap/pointcloud_to_depthimage` nodelet. Then, the mean or median depth of the object detected within the bounding boxes can be calculated. Thus, the pose of the detected objects can be calculated in real time,

tracked, and marked on a transform tree and in a costmap. Pseudocode of this process is provided in Listing 1, while Figure 8 shows the result of the detection.

Listing 1. pseudo code for generating bounding boxes.

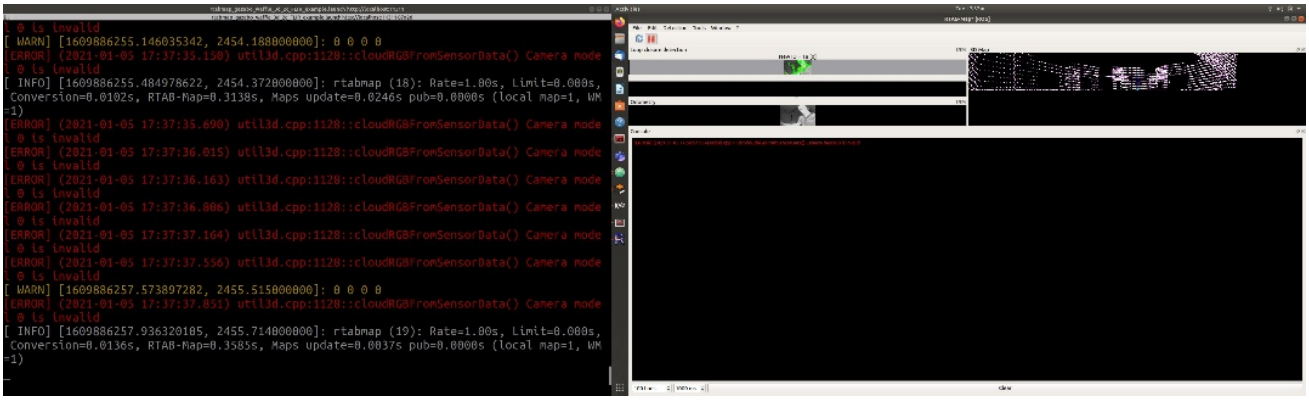
```
(xmin, xmax, ymin, ymax) = boundingBoxBoundaries(box)
depth = cv_image[((ymin+ymax)/2), ((xmin+xmax)/2)]
(x, y) = self.convertDPx2XY(depth, image) Trans-
formStamped transform;
transform.position = (x,y,depth) tfStaticBroad-
caster.sendTransform(transform, frame)
```

Figure 8. Generating bounding box around detected object in the map along with appropriate transforms in a simulation environment.



Since the FLIR camera can operate in adverse, low-visibility conditions that other RGB sensors cannot, it would be advantageous to map with the FLIR sensor. By layering a **`sensor_msgs/Image`** topic produced by **`flir_boson_usb`** with a depth or point cloud image produced by another sensor, the FLIR camera could be used to map. The mapping software RTABMap has this capability. By using the **`rtabmap/rgdb_sync`** nodelet in combination with the **`ros_imresize`** package, the scaled-up image can be layered onto a depth image or point cloud to be used for SLAM (Figure 9).

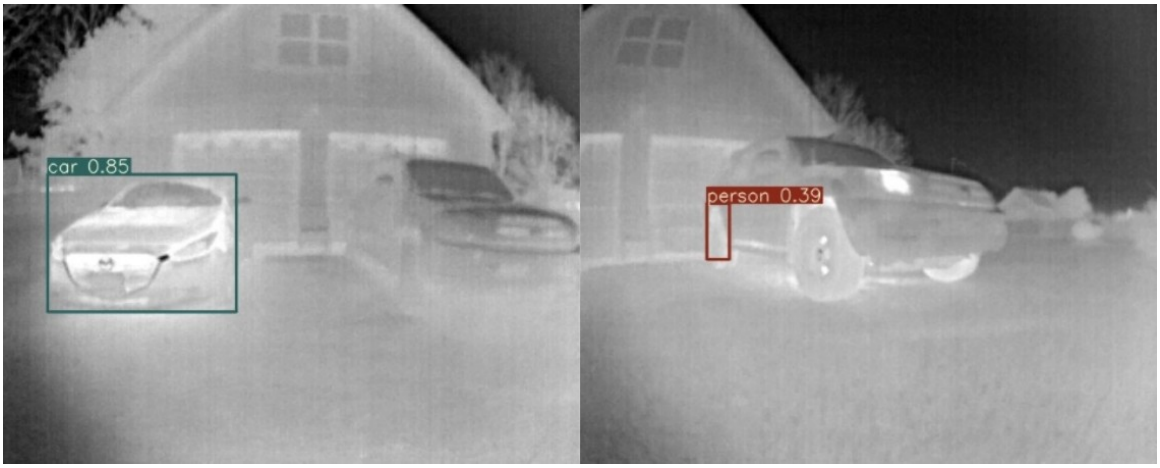
Figure 9. FLIR camera integrated into RTABMap.



3 Conclusion and Future Implementations

There are inconsistencies in the current model due to weather and environment conditions. The provided data by FLIR was recorded in the following conditions: day (60%) and night (40%) driving on Santa Barbara, California, area streets and highways during November to May with clear to overcast weather. The temperature of the surroundings affects the temperature of the object being detected, which can change the likelihood of it being recognized (Figure 8). Thus, it is recommended to generate an algorithm on objects that are in a variety of conditions. Additionally, for the algorithm to be useful in the field, more orientations of the detected objects are necessary. The FLIR dataset provided is recorded from the outside of a car, so the orientations of the car are limited. Thus, if the object is in an unusual or foreign orientation, the algorithm will not detect it (Figure 10).

Figure 10. Successful identification of car (left) and misclassification based on atmospheric conditions and orientation (*right*).



The fact that SLAM with FLIR-assisted object-detection is now possible is a milestone for search and rescue and navigation in dangerous environments, such as areas where there are hazardous material leaks, or areas where there are hostile individuals. By using `move_base_flex`, `mesh_navigation`, or another navigation package, a user could detect potentially dangerous or desirable artifacts and weigh the costmap accordingly. By using FLIR, the system can operate stealthily in the dark and fog without potentially hostile individuals knowing of its existence.

4 Installation

To install **darknet_ros**, **subt_solution_launch**, and configure YOLOv3 for ROS, follow the instructions detailed below:

Install **yolov3**:

```
$ cd && git clone https://github.com/ultralyics/yolov3.git
```

Install **darknet_ros**:

```
$ cd ~/catkin_ws/src
$ git clone -recursive https://github.com/leggedrobotics/ dark-
   net_ros.git
```

Install **subt_solution_launch**:

```
$ cd ~/catkin_ws/sr
$ git clone https://github.com/osrf/subt.git
$ git clone https://github.com/osrf/subt_hello_world.git
$ git clone https://github.com/astuff/flir_boson_usb.git
```

There may be missing dependencies for **subt** packages, if so install and try again. Some missing dependencies may be installed using the following command:

```
$ sudo apt install ignition-blueprint libignition-common3-dev libigni-
   tion-transport7-dev docker-ce docker-ce-cli ros-ign- gazebo ros-me-
   lodic-ros-ign-point-cloud libignition-gazebo3 libignition-gazebo3-
   dev ros-melodic-cartographer ros-melodic- cartographer-ros ros-me-
   lodic-rotors-control libignition-launch
   -dev ros-melodic-tf2-sensor-msgs ros-melodic-rtabmap ros- melodic-
   joy ros-melodic-teleop-twist-joy ros-melodic-teleop- twist-keyboard
   ros-melodic-laser-proc ros-melodic-rgbd-launch ros-melodic-
   depthimage-to-laserscan ros-melodic-gmapping ros- melodic-amcl ros-
   melodic-map-server ros-melodic-move-base ros- melodic-urdf ros-me-
   lodic-xacro ros-melodic-compressed-image- transport ros-melodic-
   rqt-image-view ros-melodic-navigation ros-melodic-interactive-mark-
   ers ros-melodic-velodyne-simulator
    ros-melodic-turtlebot3 ros-melodic-turtlebot3-gazebo
```

Compile and build workspace:

```
$ cd ~/catkin_ws && catkin_make -DCMAKE_BUILD_TYPE=Release -
    DCMAKE_C_COMPILER=/usr/bin/gcc-6
```

Depending on your CUDA version, you may need to change compute codes in the **CMakesList.txt** file in the **darknet_ros** package. The **darknet_ros** natively supports CUDA 8, but the version used for this report was modified to accommodate CUDA 11.

To enable CUDA11 support use the following commands:

```
$ roscd darknet_ros
$ nano CMakesList.txt
```

Replace **${CUDA_NVCC_FLAGS}** with the proper flags for your device, provided https://docs.nvidia.com/cuda/index.html. For this paper, lines 25–31 were replaced by the following:

```
-O3
-gencode arch=compute_75,code=sm_75
```

The **CUDA_DIR** variable was set manually in the **CMakesList.txt** file at line 13:

```
set(CUDA_DIR /usr/local/cuda-11.1) # or wherever CUDA toolkit is  in-
    stalled
```

It should be noted that there were issues with CUDA 11 not being detected by default. This was remedied by editing the **.bashrc** file:

```
$ sudo nano ~/.bashrc
```

Append the following to the end of the file:

```
export PATH=/usr/local/cuda-11.1/bin${PATH:+:{PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda-11.1/lib${LD_LIBRARY_PATH
    :+:{LD_LIBRARY_PATH}}
```

To install **RTABMap-ROS** and **ros-imresize** to enable FLIR SLAM with object detection integrated into the produced map, follow the instructions below:

```
$ sudo apt install ros-melodic-rtabmap-ros
$ sudo apt remove ros-melodic-rtabmap-ros
$ cd ~
$ git clone https://github.com/introlab/rtabmap.git rtabmap
$ cd rtabmap/build
$ cmake .. #[<---double dots included]
$ make
$ sudo make install
$ cd ~/catkin_ws/src
$ git clone https://github.com/introlab/rtabmap_ros.git
$ git clone https://github.com/artivis/ros_imresize.git
$ catkin_make -j1 -DRTABMAP_SYNC_MULTI_RGBD=ON - DRT-
   ABMAP_SYNC_USER_DATA=ON
```

4.1 Configuration

To configure **darknet_ros** to use custom datasets and weights for object detection, use the following instructions. These instructions assume that the custom weights have already been generated.

To link custom weights to **darknet_ros** use the following commands:

```
$ roscd darknet_ros
$ sudo ln -s /path/to/custom.weights ./yolo_network_config/weights
   /custom.weights
```

If a pytorch model (**.pt** extension) is used, convert the **.pt** file to a **.weights** file using YOLOv3:

```
$ cd ~/yolov3
$ python3 -c "from models import *; convert('path/to/configFile. cfg',
   'path/to/weightsFile.pt')"
```

Then, you will be able to link the new **.weights** file. Additionally, the **.cfg** file must also be linked:

```
$ roscd darknet_ros
$ sudo ln -s /path/to/custom.cfg ./yolo_network_config/cfg/custom. cfg
```

Then, link the **.yaml** file that determines the configuration file and object names:

```
 $ sudo ln -s /path/to/custom.yaml ./config/cus-
tom.yaml
```

Alternatively, the file can be written **/config/custom.yaml**:

```
$ nano config/cus-
tom.yaml
```

An example **custom.yaml** is provided below:

```
yolo_model:

  config_file:
    name: yolov3-tiny.cfg
  weight_file:
    name: best.weights thresh-
  old:
    value: 0.5 detection_clas-
  ses: names:
    - person
    - bicycle
    - car
    - motorcycle
    - airplane
    - bus
    - train
```

4.2 Launch

To launch yolov3 detection standalone use the following commands:

```
$ cd ~/yolov3
$ python3 detect.py --weights /path/to/weights.pt --cfg /path/to/
    conf.cfg --data /path/to/input/folder/ ---output /path/to/ out-
    put/folder/
```

To launch **darknet_ros** with **flir_boson_usb** issue the following commands:

```
$ roslaunch flir_boson_usb boson320.launch --dev /dev/video2 #[
   NOTE: --dev varies]
$ roslaunch darknet_ros custom.launch
```

To launch **rtabmap** and **imresize** to enable FLIR SLAM (assuming **darknet_ros** and **flir_boson_usb** are running) use the following commands:

```
$ roslaunch ros_imresize imresize.launch
$ roslaunch rtabmap_ros rtabmap_darknet.launch
```

5 Code Listings

To assist the reader in recreating the aforementioned setup, the complete code is provided below. The list below lists the filename and provides a short description of what it does. The launch files assume the user is using a Teledyne FLIR Boson camera.

Listing 2: custom_darknet_ros.launch—This launch file loads the `darknet_ros` node and provides paths to the weight files and various yaml files used to launch the node.

Listing 3: custom-tiny.yaml—This parameters file loads the configuration and weights file and contains a list of the detection classes used by YOLO.

Listing 4: ros_custom_params.yaml—This file sets the subscription and publishing topics used by YOLO.

Listing 5: ros_custom_rtabmap.launch—The rtabmap launch file loads all the parameters needed for SLAM and has been customized to subscribe to the topics published by `darknet_ros`.

Listing 6: resize.launch—The resize launch file resizes the camera image to a different size; specifically 1920 × 1080.

Listing 2. custom_darknet_ros.launch

```
<?xml version="1.0" encoding="utf-8"?>

<launch>

  <!-- Console launch prefix -->

  <arg name="launch_prefix" default=""/>

  <arg name="image" default="/flir_boson/image_raw"/>

  <!-- Config and weights folder. -->

  <arg name="yolo_weights_path" default="$(find dark-
net_ros)/yolo_network_config/weights"/>

  <arg name="yolo_config_path"      default="$(find dark-
net_ros)/yolo_network_config/cfg"/>



  <!-- ROS and network parameter files -->

  <arg name="ros_param_file"  default="$(find darknet_ros)/con-
fig/ros_custom_params.yaml"/>

  <arg name="network_param_file"    default="$(find dark-
net_ros)/config/custom-tiny.yaml"/>



  <!-- Load parameters -->

  <rosparam command="load" ns="darknet_ros" file="$(arg
ros_param_file)"/>

  <rosparam command="load" ns="darknet_ros" file="$(arg net-
work_param_file)"/>
```

```
  <!-- Start darknet and ros wrapper -->

  <node pkg="darknet_ros" type="darknet_ros" name="darknet_ros"
output="screen" launch-   prefix="$(arg launch_prefix)">

    <param name="weights_path" value="$(arg yolo_weights_path)"
/>

    <param name="config_path" value="$(arg yolo_config_path)" />

    <remap from="camera/rgb/image_raw” to="$(arg image)" />"

  </node>

<!--<node name="republish" type="republish" pkg="image_transport"
output="screen" args="compressed in:=/front_camera/image_raw raw
out:=/camera/image_raw" /> -->

</launch>
```

Listing 3. custom-tiny.yaml

```
yolo_model:
  config_file:
    name: yolov3-tiny.cfg
  weight_file:
    name: best.weights
  threshold:
    value: 0.5
  detection_classes:
    names:
      - person
      - bicycle
      - car
      - motorcycle
      - airplane
      - bus
      - train
      - truck
      - boat
      - traffic light
      - fire hydrant
      - stop sign
      - parking meter
```

```
- bench
- bird
- cat
- dog
- horse
- sheep
- cow
- elephant
- bear
- zebra
- giraffe
- backpack
- umbrella
- handbag
- tie
- suitcase
- frisbee
- skis
- snowboard
- sports ball
- kite
- baseball bat
- baseball glove
```

```
- skateboard
- surfboard
- tennis racket
- bottle
- wine glass
- cup
- fork
- knife
- spoon
- bowl
- banana
- apple
- sandwich
- orange
- broccoli
- carrot
- hot dog
- pizza
- donut
- cake
- chair
- couch
- potted plant
```

```
- bed
- dining table
- toilet
- tv
- laptop
- mouse
- remote
- keyboard
- cell phone
- microwave
- oven
- bike
- hydrant
- motor
- rider
- light
- sign
- motor vehicle
- human face
- hair drier
- license plate
```

Listing 4. ros_custom_params.yaml

```
subscribers:

  camera_reading:

    topic: /flir_boson/image_raw

  queue_size: 1



  actions: camera_reading:

    name: /darknet_ros/check_for_objects



publishers: object_detector:

    topic: /darknet_ros/found_object

    queue_size: 1

    latch: false



  bounding_boxes:

     topic: /darknet_ros/bounding_boxes

     queue_size: 1

     latch: false



  detection_image:

    topic: /darknet_ros/detection_
```

```
  image_queue_size: 1
  latch: true


image_view:
   enable_opencv: true
   wait_key_delay: 1
  enable_console_output: true
```

Listing 5. ros_custom_rtabmap.launch

```
 <launch>

  <arg name="gui_cfg" default="~/.ros/rtabmap_gui.ini" />

  <arg name="launch_prefix" default=""/>

  <arg name="output" default="screen"/>

  <arg name="viz" default="true"/>


  <group ns="rtabmap">

  <node pkg="nodelet" type="nodelet" name="rgbd_sync_flir" args=
"standalone rtabmap_ros/rgbd_sync" output="screen">

      <remap from="rgb/image" to="/darknet_ros/ detection_im-
age_crop"/>

      <remap from="depth/image"     to="/camera_0/depth/ im-
age_raw"/>

      <remap from="rgb/camera_info" to="/flir_boson/camera_info"
/>

      <remap from="rgbd_image"      to="rgbd_image"/> <!-- output
-->

<!-- Should be true for not synchronized camera topics (e.g.,
false for kinectv2, zed, realsense, true for xtion, kinect360)-->

      <param name="approx_sync"     value="true"/>

  </node>

  <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap" output="
screen" args="--delete_db_on_start">


      <param name="frame_id" type="string" value=" base_foot-
print"/>

      <param name="odom_frame_id" type="string" value="" />

      <param name="subscribe_depth" type="bool" value="false"/>

      <param name="subscribe_scan_cloud" type="bool" value="
true"/>

      <param name="subscribe_scan" type="bool" value="false"/>

      <param name="subscribe_odom_info" type="bool" value="
false"/>

      <!-- Add param name="subscribe_odom_info" type="bool"
value="true"/> to fix problems with rtabmap subscribing to
odom_info -->
```

```
      <param name="subscribe_rgb"   type="bool" value=" false"/>

      <param name="subscribe_rgbd"  type="bool" value=" true"/>

      <param name="rgbd_cameras"    type="int"  value="1"/>

      <remap from="rgbd_image"      to="rgbd_image"/>

      <remap from="odom" to="/odom"/>

      <remap from="scan_cloud" to="/velodyne_points"/>

      <param name="queue_size" type="int" value="10"/>

      <param name="RGBD/NeighborLinkRefining" type="string"
value="true"/> <!-- Do odometry correction with consecutive laser
scans -->

      <param name="RGBD/ProximityBySpace" type="string"
value="true"/> <!-- Local loop closure detection (using es-ti-
mated position) with locations in WM -->

      <param name="RGBD/ProximityByTime"  type="string"
value="false"/> <!-- Local loop closure detection with loca-tions
in STM -->

      <param name="RGBD/ProximityPathMaxNeighbors" type=" string"
value="10"/> <!-- Do also proximity detection by space

by merging close scans together. -->

      <param name="RGBD/OptimizeFromGraphEnd" type="string"
value="false"/> <!-- Optimize graph from initial node so map to
odom transform will be generated -->

      <param name="RGBD/OptimizeMaxError" type="string"
value="3"/> <!-- Reject any loop closure causing large errors
(>3x link's covariance) in the map -->

      <param name="RGBD/LocalRadius"      type="string"
value="5"/> <!-- limit length of proximity detections -->

      <param name="Reg/Strategy"    type="string" value="1"/>
      <!-- 0=Visual, 1=ICP, 2=Visual+ICP -->

      <param name="Reg/Force3DoF"   type="string" value="true"/>
<!-- 2D SLAM -->

      <param name="Grid/FromDepth"  type="string" value="false"/>
<!-- Create 2D occupancy grid from laser scan -->

      <param name="Mem/STMSize"     type="string" value="30"/>
      <!-- increased to 30 to avoid adding too many loop closures
on just seen locations -->

      <param name="Vis/MinInliers"  type="string" value="15"/>
      <!-- 3D visual words correspondence distance -->

      <param name="Kp/DetectorStrategy"   type="string"
value="10"/>
```

```
      <param name="Vis/FeatureType" type="string" value="10"/>


      <!-- ICP parameters -->

      <param name="Icp/VoxelSize"   type="string " value="0.01"/>

      </node>


      <!-- Visualisation RTAB-Map -->

  <node if="$(arg viz)" pkg="rtabmap_ros" type="rtabmapviz"
name="rtabmapviz" args="-d $(arg gui_cfg)" output="$(arg output)"
launch-prefix="$(arg launch_prefix)">

      <param name="subscribe_rgb"   type="bool" value=" false"/>

      <param name="subscribe_rgbd"  type="bool" value=" true"/>

      <param name="subscribe_odom_info" type="bool" value="
false"/>

      <param name="subscribe_scan_cloud" type="bool" value="
true"/>

      <param name="subscribe_scan" type="bool" value="false"/>

      <param name="subscribe_depth" type="bool" value="false "/>

      <param name="frame_id" type="string" value="base_foot-
print"/>

      <param name="odom_frame_id" type="string" value="”/>

      <param name="queue_size" type="int" value="10"/>

      <param name="rgbd_cameras" type="int" value="1"/>

      <remap from="rgbd_image" to="rgbd_image"/>

      <remap from="odom"      to="/odom"/>

      <remap from="scan_cloud" to="/velodyne_points"/>

      </node>


      <node pkg="tf" type="static_transform_publisher" name="
foot_to_rgb_0" args="0.064 -0.065 0.094 -1.57 0 -1.57 /
base_footprint /camera_0_rgb_optical_frame 100" />

      <node pkg="tf" type="static_transform_publisher" name="
foot_to_depth_0" args="0.064 -0.065 0.094 -1.57 0 -1.57 /
base_footprint /camera_0_depth_optical_frame 100" />

      <node pkg="tf" type="static_transform_publisher" name="
foot_to_rgb_1" args="-0.2 -0.065 0.094 1.57 0 -1.57 / base_foot-
print /camera_1_rgb_optical_frame 100" />
```

```
      <node pkg="tf" type="static_transform_publisher" name="
foot_to_depth_1" args="-0.2 -0.065 0.094 1.57 0 -1.57 /
base_footprint /camera_1_depth_optical_frame 100" />

      <node pkg="tf" type="static_transform_publisher" name="
foot_to_scan" args="-0.064 0 0.122 0 0 0 /base_footprint /

      base_scan 100" />

      <node pkg="tf" type="static_transform_publisher" name="
foot_to_velodyne" args="0 0 0.44 0 0 0 /base_footprint /

      velodyne 100" />

      <node pkg="tf" type="static_transform_publisher" name="
depth_0_to_detection_image" args="0 0 0 0 0 0 / cam-
era_0_depth_optical_frame /detection_image 100" />

  </group>

</launch>
```

Listing 6. resize.launch

```
<launch>

  <arg name="width" default="1920" />

  <arg name="height" default="1080" />



  <arg name="camera_info" default="/flir_boson/camera_info" />

  <arg name="camera_topic" default="/darknet_ros/detection_image"
/>

  <arg name="undistord" default="false"/>



  <node name="ros_imresize" pkg="ros_imresize" type="ros_im-
resize" output="screen">

<param name="camera_info" value="$(arg camera_info)" />

<param name="topic_crop" value="$(arg camera_topic)" />

<param name="resize_width" value="$(arg width)" />

<param name="resize_height" value="$(arg height)" />

<param name="undistord" value="$(arg undistord)" />

  </node>

</launch>
```

References

Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. "YOLOv4: Optimal Speed and Accuracy of Object Detection." *arXiv:2004.10934*. https://arxiv.org/abs/2004.10934v1.

Girshick, Ross. 2015. "Fast R-CNN." In *Proceedings of the IEEE International Conference on Computer Vision*, 11–18 December 2015, Santiago, Chile, 1440–1448. https://doi.org/10.1109/ICCV.2015.169.

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 23–28 June 2014, Columbus, Ohio, 580–587.

Glaspell, Garry P., Steven R. Lessard, Benjamin A. Christie, Kyle Jannak-Huang, Noah C. Wilde, Weiyu He, Osama Ennasr, Daniel T. Pham, Daniel B. Hasemann, Philip M. Devine, and John Kiene. 2020. *Optimized Low Size, Weight, Power and Cost (SWaP-C) Payload for Mapping Interiors and Subterranean on an Unmanned Ground Vehicle*. ERDC/GRL TR-20-6. Alexandria, VA: U.S. Engineer Research and Development Center, Geospatial Research Laboratory. https://hdl.handle.net/11681/35878.

Labbé, Mathieu and François Michaud. 2019. "RTAB-Map as an Open-Source Lidar and Visual Simultaneous Localization and Mapping Library for Large-Scale and Long-Term Online Operation." *Journal of Field Robotics* 36, no. 2 (March): 416–446. https://doi.org/10.1002/rob.21831.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You Only Look Once: Unified, Real-Time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 26 June–1 July 2016, Las Vegas, Nevada, 779–788. https://doi.org/10.1109/CVPR.2016.91.

Redmon, Joseph and Ali Farhadi. 2017. "YOLO9000: Better, Faster, Stronger." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 21–26 July 2017, Honolulu, Hawaii, 6517–6526. https://doi.org/10.1109/CVPR.2017.690.

Redmon, Joseph and Ali Farhadi. 2018. "YOLOv3: An Incremental Improvement," *arXiv:1804.02767*. https://arxiv.org/abs/1804.02767.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2017. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, no. 6 (June): 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031.

Teledyne FLIR. n.d. "FREE FLIR Thermal Dataset for Algorithm Training." Accessed 31 May 2021. https://www.flir.com/oem/adas/adas-dataset-form/.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2021	Final Report	

4. TITLE AND SUBTITLE
ROS Integrated Object Detection for SLAM in Unknown, Low-Visibility Environments

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT
622146

6. AUTHOR(S)
Benjamin Christie, Osama Ennasr, Garry Glaspell

5d. PROJECT NUMBER
AT9

5e. TASK NUMBER
01

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center (ERDC)
Geospatial Research Laboratory (GRL)
7701 Telegraph Road
Alexandria, VA 22315-3864

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDC/GRL TR-21-6

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Corps of Engineers
Washington, DC 20314-1000

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Integrating thermal (or infrared) imagery on a robotics platform allows it to function in low-visibility environments, such as pure darkness or low-density smoke. To maximize the effectiveness of this approach we discuss the modifications required to integrate our low-visibility object detection model on the Robot Operating System (ROS). Furthermore, we introduce a method for reporting detected objects while performing Simultaneous Localization and Mapping (SLAM) by generating bounding boxes and their respective transforms in visually challenging environments.

15. SUBJECT TERMS
Automated vehicles; Vehicles, Military; Visibility; Optics; Infrared imaging

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)
Unclassified	Unclassified	Unclassified	SAR	39	