

# The Sick LIDAR Matlab/C++ Toolbox: Software for Rapidly Interfacing/Configuring Sick LIDARs

Jason C. Derenick<sup>†</sup>, Thomas H. Miller<sup>†</sup>, John R. Spletzer<sup>†</sup>,  
Aleksandr Kushleyev<sup>‡</sup>, Tully Foote<sup>‡</sup>, Alex Stewart<sup>‡</sup>, Jon Bohren<sup>‡</sup> and Daniel D. Lee<sup>‡</sup>

**Abstract**—The Sick LIDAR Matlab/C++ Toolbox is an open-source project that provides a stable and easy-to-use Matlab/C++ interface for both Sick LMS 2xx and Sick LD laser range finders. In addition to multi-threaded device drivers for Linux, the package features a mex (Matlab executable) interface that allows the end-user to stream real-time range and reflectivity data directly into Matlab. This feature is especially attractive as it facilitates the rapid development of algorithms by exploiting the high-level functionality afforded by Matlab’s vector operations. It is bundled with a collection of examples – including utilities for easily reconfiguring both LMS and LD models. Ample documentation is provided. As the toolbox is branched from the source code used by the Ben Franklin Racing Team during the DARPA Urban Challenge (DUC), we present examples highlighting its utility in this context. Our entry – Little Ben – was one of only six vehicles to successfully complete the 55 mile Urban Challenge Final Event (UFE).

## I. INTRODUCTION & RELATED WORK

Introduced in early 1990’s, Sick non-contact range measurement systems have become ubiquitous in the robotics community. An industrial-grade sensor, they have been utilized in a wide variety of applications. These include, but are by no means limited to, autonomous wheelchair systems for assisted living [1], urban reconnaissance for military applications [2], simultaneous localization and mapping (SLAM) [3], [4] including volumetric environment mapping [5]–[7], robotic tour-guides [8], and autonomous road-vehicle systems [9]–[11]. In fact, all eleven finalists in the recent DARPA Urban Challenge chose Sick LMS 2xx and/or Sick LD laser range finders to include in their sensor suite.

Despite the overwhelming popularity of the Sick sensors – attributed to their reliability, accuracy, range, and reasonable cost – until this point the robotics community has been lacking an open-source software package designed specifically for their use. Many opt to use a basic implementation that is bundled in the context of an auxiliary framework such as Player/Stage [12], which may not be well-suited for the

This work was funded in part by Thales Communications, Inc., the Commonwealth of Pennsylvania, Department of Community and Economic Development, and by the National Science Foundation Partnerships for Innovation (PFI) Program under Grant No. 0650115. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

<sup>†</sup> Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, 18015 USA {derenick, thm204, josa}@lehigh.edu

<sup>‡</sup> GRASP Laboratory, University of Pennsylvania, Philadelphia, PA, 19104 USA {akushley, tfoote, alexad, jbohren, ddlee}@seas.upenn.edu



Fig. 1. The toolbox is branched from the source code used by the Ben Franklin Racing Team during the DARPA Urban Challenge. Our entry – Little Ben – employed five Sick LMS 291 and LD-LRS LIDARs during the race and was one of only six vehicles to successfully complete the 55 mile Urban Challenge Final Event. Ben ran on fewer than 5,000 lines of Matlab script – using mex interfaces to device drivers and messaging.

desired application. As a result, there are also a few open-source drivers; however, at best they provide little more than the ability to grab data from the device and process it using C/C++, offering minimal configuration capabilities [13], [14]. Additionally, no publicly available framework (nor driver) currently offers support for the long-range Sick LD units, and none support a high-level scripting interface for either Sick family. As a result, writing code to process data is often non-trivial and requires familiarity with C/C++.

In contrast to these efforts, the Sick LIDAR Matlab/C++ Toolbox takes a holistic view of interfacing to the Sick LMS 2xx and Sick LD families of laser range finders. The focus is on making data processing as simple as possible while keeping the high-level interface clean and concise. In so doing, it provides Matlab mex interfaces and easy-to-use (but well-featured) C++ device drivers for data acquisition and configuration under Linux OS. Together these elements allow range and reflectivity data to be streamed directly into Matlab at full data rate. At first glance, this may seem like a secondary contribution; however, we cannot over-emphasize the importance of this feature as it *facilitates the rapid development* of algorithms by exploiting the high-level functionality afforded by Matlab’s vector operations.

Motivating the use of Matlab, we note that the toolbox is branched from the source code used by the Ben Franklin Racing Team during the DARPA Urban Challenge. Our entry – Little Ben – employed five Sick LMS 291 and LD-LRS LIDARs during the race and was one of only six vehicles to successfully complete the 55 mile Urban Challenge Final Event. Little Ben successfully completed the UFE using fewer than 5,000 lines of Matlab script – using mex interfaces

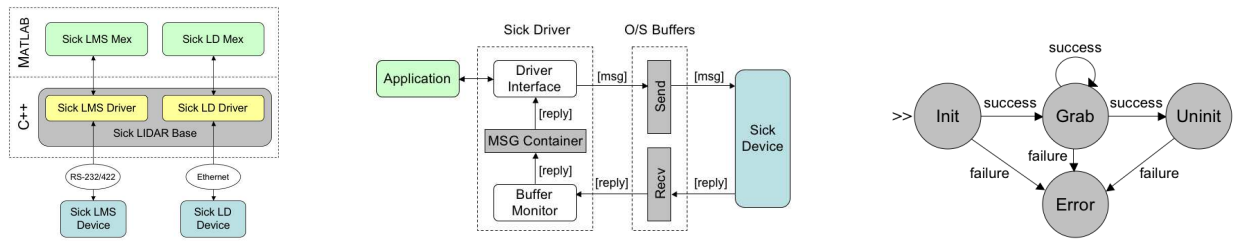


Fig. 2. (Left) The high-level schematic for the Sick LIDAR Matlab/C++ Toolbox. High-level mex functions directly interact with C++ drivers that share a common base framework. (Center) Schematic of a typical application using the interface provided by the low-level driver. Each driver features a buffer monitor thread that is responsible for ensuring the receive buffer never becomes saturated. The most recent Sick LMS reply is buffered in a message container where the interface can access it. (Right) The quaternary state machine implemented by Matlab/C++ programs using the toolbox to acquire data.

to low-level device drivers and messaging facilities. This was one to two orders of magnitude less than some other finalists, and it highlights the potential of Matlab as a run-time environment for real-time, experimental robotics.

## II. TOOLBOX FEATURES

In this section, we present a more detailed outline of the features offered by the toolbox regarding each Sick family.

### A. Sick LMS 2xx Support

For the Sick LMS 2xx models, the toolbox provides/supports the following features:

- 1) Multi-threaded (POSIX) C++ RS-232/422 driver:
  - a) Supports 500Kbps baud via a USB-COMi-M adapter
  - b) Auto-detects Sick LMS 2xx baud rate
  - c) Supports a variety of data streams, including:
    - i) High-resolution partial/interlaced scans
    - ii) Mean-measured value scans
    - iii) Measured value subrange scans
    - iv) Mean measured-value subrange scans
    - v) Range with reflectivity values concurrently (only LMS Fast, *i.e.* LMS 211/221/291-S14)
    - vi) Reflectivity-only (non LMS fast models)
  - d) Stream data with (or without) real-time indices
  - e) Advanced configuration via device driver:
    - i) Variant  $\{180^\circ/0.5^\circ, 100^\circ/0.25^\circ\}$  – *i.e.* scan resolution and field-of-view
    - ii) Measuring mode
    - iii) Availability level (useful for enabling dazzle recovery)
    - iv) Measuring units  $\{cm, mm\}$
    - v) Sensitivity  $\{high, standard, medium, low\}$
- 2) Configure (and view) device parameters via *lms\_config* utility (operates using said driver)
- 3) Matlab mex interface (*i.e.* *sicklms*):
  - a) Stream range and/or reflectivity returns
  - b) Set device variant
  - c) Display parameters

### B. Sick LD Support

Regarding the Sick LD family of laser range finders, the toolbox provides/supports the following features:

### 1) Multi-threaded (POSIX) C++ Ethernet driver:

- a) Stream range with (or without) reflectivity returns
- b) Supports multiple scan sectors
- c) Advanced configuration via device driver:
  - i) Motor speed (5–20Hz)
  - ii) Resolution (up to  $1^\circ$  in multiples of  $0.125^\circ$ )
  - iii) Multiple active/measuring scan sectors (a maximum of four)
  - iv) Temporary scan sectors (does not require write to flash)
  - v) Absolute and relative time
  - vi) Device signals (including LEDS)
  - vii) Nearfield suppression
  - viii) Sensor ID

### 2) Configure (and view) basic device parameters via *ld\_config* utility (operates using said driver).

### 3) Matlab mex interface (*i.e.* *sickld*):

- a) Stream range with (or without) reflectivity returns from multiple sectors
- b) Display configuration parameters.

Additionally, the toolbox provides a quick start manual as well as numerous examples illustrating the various ways in which the C++/Matlab interfaces can be used. A tutorial for enabling 500Kbps baud for Sick LMS 2xx models via a USB-COMi-M industrial adapter is also included. All C++ code is commented using Doxygen, and the package was written to exploit the capabilities of GNU Autotools for portability across multiple platforms.

## III. ARCHITECTURAL OVERVIEW

Figure 2 (Left) shows a high-level schematic of the Sick LIDAR Matlab/C++ Toolbox. At the highest level, the toolbox is comprised of a set of Matlab mex files that directly interact with a corresponding low-level C++ driver. All drivers are comprised of two fundamental components that are derived from a common base. In particular, each driver features an interface to acquire data and configure the device as well as a buffer monitor thread responsible for ensuring only the most recent scan information is returned via said interface. As expected, the interface is used to send commands to the device. All replies are handled by the buffer monitor where they are encapsulated as a message object

before being stored in the container where the interface can acquire them. This process is illustrated in Figure 2 (Center).

#### IV. APPLYING THE TOOLBOX

Using the toolbox to acquire data is straightforward requiring only three separate function calls: one to initialize the device, one to grab the measurements and one to uninitialized the device. This approach is similar with both the C++ and Matlab interfaces. Any application utilizing the toolbox to acquire data should follow the quaternary state machine presented in Figure 2 (Right). To highlight the elegance of the toolbox, we now focus on its high-level Matlab capabilities referring the reader to the quick start manual and examples for specifics regarding the C++ interface.

The toolbox provides two high-level interfaces: *sicklms* and *sickld*. Each provides access to the most useful functionality of its corresponding low-level C++ driver.

##### A. Interfacing the Sick LMS 2xx

The *sicklms* mex interface can be invoked from within Matlab as follows:

```
sicklms(CMD, ARGS)
```

where *CMD* denotes a string representing the command to issue the device via the low-level driver and *ARGS* denotes any arguments that the command may take.

*a) Supported Commands:* For our purposes, the toolbox was designed to support the commands provided in Table I. Specifically, the interface provides a means to initialize the device and grab whichever type of data the device may be configured to stream. To provide quick access to the current configuration parameters, we also include the *info* command which will display such settings as current measuring mode, measuring units, availability level, etc..

CMD	ARGS
init	DEVICE_PATH, DESIRED_BAUD
variant	SCAN_ANG, SCAN_RES
grab	N/A
info	N/A

TABLE I

COMMANDS SUPPORTED BY THE *sicklms* MEX INTERFACE.

*b) Return Values:* All calls that return values will return a Matlab structure. Currently, only *init* and *grab* will return structures upon success. In particular, *init* will return a three element structure as defined in Table II. The information returned from *init* can be used for properly setting up the calling m-file script. For instance, in the example *lms\_stream*, *meas\_mode* is used to properly determine the overflow values to filter invalid measurements.

Additionally, calling *grab* via *sicklms* returns a four element structure having the elements in Table III. Note that the structure contains two vectors: one to hold the current range values (*i.e.* *range*) and one to hold the corresponding reflectivity (*i.e.* *reflect*). If the device is an LMS Fast

Field	Description
<i>lms_fast</i>	(Boolean) True if device is LMS Fast, false otherwise
<i>units_mm</i>	(Boolean) True if units are mm, false if units are cm
<i>meas_mode</i>	Current measurement mode of device

TABLE II

STRUCTURE RETURNED FROM INIT COMMAND VIA *sicklms*

then both vectors will be populated with the most recent scan data. If the device is not an LMS Fast, it will populate either *range* or *reflect* depending upon what the device is configured to return (this is a function of the device measuring mode which can be set using *lms\_config*).

Field	Description
<i>res</i>	Angular scan resolution (0.25°, 0.50°, or 1.0°)
<i>fov</i>	Angular scan angle/fov (90°, 100°, or 180°)
<i>range</i>	$n \times 1$ vector of range values
<i>reflect</i>	$n \times 1$ vector of reflectivity values

TABLE III

STRUCTURE RETURNED FROM GRAB COMMAND VIA *sicklms*

*c) Examples:* Figure 4 shows a typical m-file implementation utilizing *sicklms*. Notice the correspondence with the state machine given in Figure 2 (Right). Additionally, Figure 3 (Left) shows a Matlab plot generated from streaming range and reflectivity data from a Sick LMS 291-S14 via the *sicklms* mex interface. The two peaks in reflectivity correspond with two retro-reflectors positioned within the scan area. For more usage examples, be sure to see the example m-files (*i.e.* *lms\_cart*, *lms\_stream*, *lms\_variant*) included with the toolbox as well as the quick start guide.

```

1  try
2
3      % Initialize the Sick LMS 2xx
4      ret=sicklms('init','/dev/ttyUSB0',500000);
5      disp(ret);
6
7      % Grab some measurements
8      for i=1:10
9          ret=sicklms('grab');
10         disp(ret);
11     end
12
13     % Uninitialize the device
14     clear sicklms;
15
16 catch
17     error('An error occurred!');
18 end

```

Fig. 4. A simple example of using the Sick LMS 2xx mex interface provided by the Sick LIDAR Matlab/C++ Toolbox. It implements the quaternary state machine presented in Figure 2 (Right). In this case, the Sick LMS was connected via a USB-COMi-M adapter accessible via */dev/ttyUSB0*. The requested baud rate was the device maximum at 500Kbps.

##### B. Interfacing the Sick LD

In a similar fashion, the *sickld* mex interface can be invoked from within Matlab as follows:

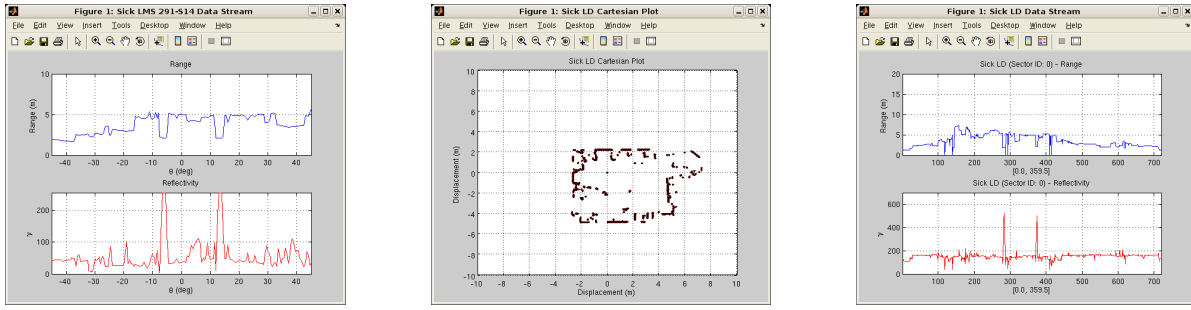


Fig. 3. (Left) Range and reflectivity values obtained from a Sick LMS 291–S14 at full data rate via the *sicklms* device interface. The two peaks in reflectivity correspond to two separate retro-reflectors within the scan area of the device. (Center) A Cartesian map of the VADER Laboratory generated from range values returned from a Sick LD–LRS 1000 scanning  $360^\circ$  with  $0.5^\circ$  resolution. The map was generated using the included *ld\_cart* example. Values were obtained using the *sicklms* mex interface. (Right) Range and reflectivity values from a Sick LD–LRS 1000 scanning with the same parameters as (Center). In this case, the two peaks in reflectivity correspond to two retro-reflectors placed within the scan area of the device.

`sicklms(CMD, ARGS)`

where `CMD` and `ARGS` are as previously defined.

*d) Supported Commands:* This interface follows the same usage pattern as *sicklms* requiring the m-file to first initialize the device, grab data and then uninitialized it. The supported commands are presented in Table IV.

CMD	ARGS
<code>init</code>	<code>DEVICE_IP</code>
<code>range</code>	N/A
<code>range+reflect</code>	N/A
<code>info</code>	N/A

TABLE IV

COMMANDS SUPPORTED BY THE *sicklms* MEX INTERFACE.

*e) Return Values:* In contrast to the Sick LMS 2xx models, the Sick LD operates by scanning user-defined active/measuring sectors – each defined by mutually exclusive subareas of its maximum  $360^\circ$  field-of-view. Additionally, all LD models support the concurrent streaming of reflectivity values with the range returns without having to reconfigure the flash on the device. As a result, we supplant the `grab` command (seen with *sicklms*) with two separate commands: `range` and `range+reflect`. They serve to specify the type of data stream to request from the device. As the device can also return data from multiple sectors, both will return an array of structures with each structure corresponding to the data obtained/associated with a given sector. Table V shows the associated members.

Field	Description
<code>id</code>	Sector ID
<code>res_ang</code>	Angular resolution over sector
<code>beg_ang</code>	Beginning angle of sector (included in scan)
<code>end_ang</code>	Ending angle of sector (included in scan)
<code>beg_time</code>	Time at which first measurement was obtained
<code>end_time</code>	Time at which last measurement was obtained
<code>range</code>	$n \times 1$ vector of range measurements
<code>reflect</code>	$n \times 1$ vector of reflectivity Measurements

TABLE V

STRUCTURE FOR A SINGLE SECTOR RETURNED FROM *sicklms*

*f) Examples:* Figure 5 shows a simple m-file implementation where the sector ID is displayed for each sector. Each call to *sicklms* using `range+reflect` will return the most recently buffered data for each. Additionally, Figure 3 (Center,Right) shows the graphical output of two examples included with the toolbox. Figure 3 (Center) shows a Cartesian map of the VADER Laboratory at Lehigh University. It was constructed using the *ld\_cart* example that is included in the toolbox. Figure 3 (Right) shows the range and reflectivity returns via the *ld\_stream* example for a single measuring sector. Both figures were generated using a Sick LD–LRS 1000 scanning  $360^\circ$  at  $0.5^\circ$  resolution. For additional examples, see the quick start manual included with the package.

```

1  try
2
3      % Initialize the Sick LD
4      sicklms('init', '192.168.0.12');
5
6      for i=1:10
7
8          % Grab the most recent values
9          ret=sicklms('range+reflect');
10
11         % Print the ID of each sector
12         for j=1:length(ret)
13             disp(ret(j).id);
14         end
15     end
16
17     % Uninitialize the device
18     clear sicklms;
19
20
21 catch
22     error('An error occurred!');
23 end

```

Fig. 5. A simple example of using the *sicklms* interface bundled with the Sick LIDAR Matlab/C++ Toolbox. In this case, the Sick LD was asked to stream both range and reflectivity information from each active/measuring sector.

## V. AT THE DARPA URBAN CHALLENGE

To illustrate the utility of the toolbox, we present it in the context of our DARPA Urban Challenge entry – highlighting

its application to 1-D terrain classification and traffic line detection. In particular, we detail two of the algorithms utilized to process Sick LMS 291-S14 range and reflectivity returns and then present graphical output of the Matlab process that was run on Little Ben during the DUC/UFE. In so doing, we also implicitly motivate the use of Matlab as a high-level run-time for experimental robotics.

### A. Terrain Classification

To detect obstacles, our vehicle employed a two-phased approach. During the first phase, the range returns from the Sick LMS 291-S14 LIDARs were mapped to  $z$  values by making the standard assumption of a locally-flat ground plane. Employing an Iteratively Re-weighted Least Squares (IRLS) algorithm [15], a linear fit is applied to these points yielding an estimate of the believed ground surface. To thwart ground plane drift, a Tikhonov-type regularization term can be included in the formulation.

For the second phase, the task becomes to classify the scan points using in part the extracted ground plane from Phase I. To this end, we consider a simple ternary classifier, aimed at assigning points one of three labels: road ( $\mathcal{G}$ ), obstacle ( $\mathcal{O}$ ), or unknown ( $\mathcal{U}$ ). The design of our classifier was based in traditional Bayesian techniques. In our implementation, we chose to bin the points into groups of four (due to the resolution of our cost map) and then characterize these bins by two distinct features: the maximal normal distance to the extracted ground plane ( $d_{B_i}$ ), and the maximal difference in  $z$  values among all of the points ( $\delta z_{B_i}$ ) – where  $B_i$  denotes the  $i^{th}$  bin. The first feature captures the *elevation* of the points while the second captures their *smoothness*.

Observing  $X = [d_{B_i} \ \delta z_{B_i}]^T$  is a discrete random variable, we model  $P(X|\mathcal{G})$  as a zero mean Gaussian (*i.e.*  $X \sim \mathcal{N}(0, \Sigma)$  where  $\Sigma = \begin{bmatrix} \sigma_d & 0 \\ 0 & \sigma_{\delta z} \end{bmatrix}$ ) and define the policy:

$$\mathcal{P}(d_{B_i}, \delta z_{B_i}) \equiv \begin{cases} B_i \in \mathcal{G}, & (d_{B_i} < 3\sigma_d) \wedge (\delta z_{B_i} < 3\sigma_{\delta z}) \\ B_i \in \mathcal{O}, & (d_{B_i} \geq 3\sigma_d) \vee (\delta z_{B_i} \geq 3\sigma_{\delta z}) \\ B_i \in \mathcal{U}, & \text{otherwise} \end{cases}$$

Here we generate the thresholds for the three classes based on the single model used to characterize safe terrain. In practice, it yielded highly satisfactory results.

Figure 6 shows the Matlab process output for the 1-D terrain classification algorithm run on Little Ben. In practice, we chose  $\sigma_d = 5$  cm and  $\sigma_{\delta z} = 3$  cm.

### B. Traffic Line Detection

To mitigate any positional bias due to shifts in GPS, we also implemented a Matlab process to extract points corresponding to traffic lines using the reflectivity returns of the Sick LMS 291-S14. Assuming that a standard line would be  $\approx 10$ – $15$ cm, it was guaranteed that any traffic line would incur at least two (no more than three) direct hits by the LIDAR during each scan. Moreover, assuming a standard lane width of  $\approx 4$ m, we reduced the feasible set of values to include only those within  $\pm 2.25$ m from vehicle center.

Given these observations, we began by mapping the feasible set (denoted  $\mathcal{R}$ ) to a binary representation using adaptive

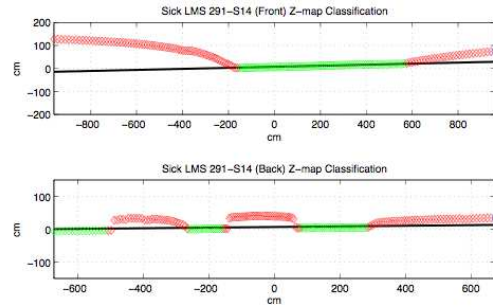


Fig. 6. Classification results of the 1-D obstacle detection (Matlab) process running on our DUC/UFE entry, Little Ben. Note that the points are classified based upon the label assigned to their respective bins with green denoting  $\mathcal{G}$ , red denoting  $\mathcal{O}$ , and blue denoting  $\mathcal{U}$ .

thresholding based upon its mean reflectivity ( $\mu_{\mathcal{R}}$ ) using the threshold  $\tau^{min}$ . This value represents the minimal difference in reflectivity from  $\mu_{\mathcal{R}}$  that is required for a scan point to be considered a possible traffic line hit. Next, we eliminate any reflectivity values that correspond to points classified as either  $\mathcal{O}$  or  $\mathcal{U}$  and collect the remaining points into regions of interest (ROIs). For our purposes, a ROI is defined as two or three consecutive scan points with feasible reflectivity values. All ROIs are considered likely candidates for traffic line hits.

In an effort to ensure robustness, we enforce the rule that if there is any ambiguity in a result then to ignore the data. This includes ignoring results that suggest more than two lines on either side of vehicle center. This is a simple heuristic as most traffic lines are laid in single or pair.

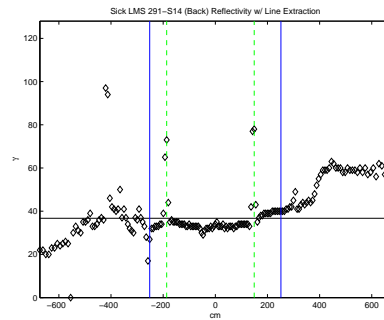


Fig. 7. Classification results of the 1-D traffic line detection (Matlab) process running on Little Ben. The two peaks (denoted with vertical green lines) correspond to two separate traffic lines on either side of the vehicle. The blue lines bound the feasible set of reflectivity values.

Figure 7 shows the Matlab process output of our traffic line extraction algorithm run on our DUC entry. In this example, a single traffic line hit is detected on either side of the vehicle. In practice, we chose  $\tau_{\gamma}^{min} = 10$ . This enabled properly classifying everything from faded yellow lines to freshly placed, white traffic tape.

## VI. ACQUIRING THE TOOLBOX

The toolbox can be downloaded from:

<http://vader.cse.lehigh.edu/sicktoolbox>

## VII. DISCUSSION AND FUTURE WORK

In this paper, we presented the Sick LIDAR Matlab/C++ Toolbox – branched from the source code used by one of only six vehicles to successfully complete the DARPA Urban Challenge. The toolbox is designed emphasizing easy data acquisition from both Sick LMS 2xx and Sick LD laser range finders. It provides both well–featured, multi–threaded C++ drivers as well as a high–level Mex interface for streaming data directly into Matlab and is released under the GNU General Public License. In addition to source code, the toolbox also provides a step–by–step tutorial for enabling 500Kbps baud using a Sick LMS 2xx interfaced with a USB–COMi–M adapter. A quick start manual is also included.

It was also presented in the context of the DARPA Urban Grand Challenge for our entry – Little Ben. The examples include details regarding the teams implementation for 1–D terrain classification and traffic line detection.

There are obvious ways in which the toolbox can be expanded and or improved. For instance, it is currently available to work only under Linux OS. A version for Windows would be a next obvious step. Additionally, efforts are underway to expand the scope of toolbox to beyond just Sick LIDAR support. In particular, we are currently looking to generalize the toolbox by providing Hokuyo and Velodyne – both of which were also featured in Little Ben’s sensor suite. Like Sick LIDARs, both are capable of providing range and reflectivity information.

## REFERENCES

- [1] C. Gao, I. Hoffman, T. Panzarella, and J. Spletzer, “Automated transport and retrieval system (atrs): A technology solution to automobility for wheelchair users,” in *Proceedings of the 2007 International Conference on Field and Service Robotics (FSR '07)*, Chamonix, France, July 2007.
- [2] B. Yamauchi, “Autonomous urban reconnaissance using man portable ugv’s,” in *Proceedings of Unmanned Systems Technology VIII (SPIE Conference 6230)*, 2006.
- [3] S. Thrun, W. Burgard, and D. Fox, “A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping,” *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 321–328 vol.1, 2000.
- [4] P. Newman, D. Cole, and K. Ho, “Outdoor slam using visual appearance and laser ranging,” *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1180–1187, May 15–19, 2006.
- [5] C.-C. Wang and C. Thorpe, “A hierarchical object based representation for simultaneous localization and mapping,” in *Proceedings of the IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems (IROS '04)*, Sendai, Japan, September–October 2004.
- [6] P. Skrzypczynski, “Spatial uncertainty management for simultaneous localization and mapping,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '07)*, Roma, Italy, April 2007.
- [7] M. Montemerlo, D. Hahnel, D. Ferguson, R. Triebel, W. Burgard, S. Thayer, W. Whittaker, and S. Thrun, “A system for three-dimensional robotic mapping of underground mines,” Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-02-185, October 2002.
- [8] R. Philippsen and R. Siegwart, “Smooth and efficient obstacle avoidance for a tour guide robot,” *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 1, pp. 446–451, 14–19 Sept. 2003.
- [9] “Tartan Racing Team: Boss at a glance.” [Online]. Available: <http://www.tartanracing.org/press/boss-glance.pdf>
- [10] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niek-erk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kachler, A. Nefian, and P. Mahoney, “Winning the DARPA Grand Challenge,” *Journal of Field Robotics*, 2006, accepted for publication.
- [11] “2007 darpa urban challenge: The ben franklin racing team b156 technical paper.” [Online]. Available: [http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Ben\\_Franklin\\_Driving.pdf](http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Ben_Franklin_Driving.pdf)
- [12] B. Gerkey, R. Vaughan, and A. Howard, “The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems,” 2003.
- [13] “lmsapi.” [Online]. Available: <http://lmsapi.sourceforge.net/>
- [14] D. Yuen, “Lms200demo.” [Online]. Available: <http://www.ele.auckland.ac.nz/cyue001/tech/LMSintro.html>
- [15] A. Guitton, “Stanford lecture notes: The iteratively reweighted least squares method,” 2000.