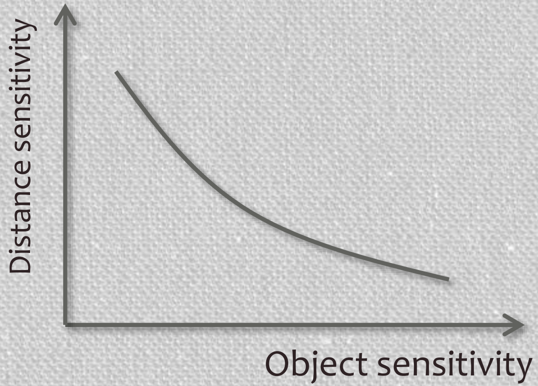
APPLICATION OF MATLAB & CODER FOR AN AUTOMOTIVE VISION PROOF OF CONCEPT

Maung Han
Alpine Electronics Research
May 2016

Outline

Why Detect Ground?

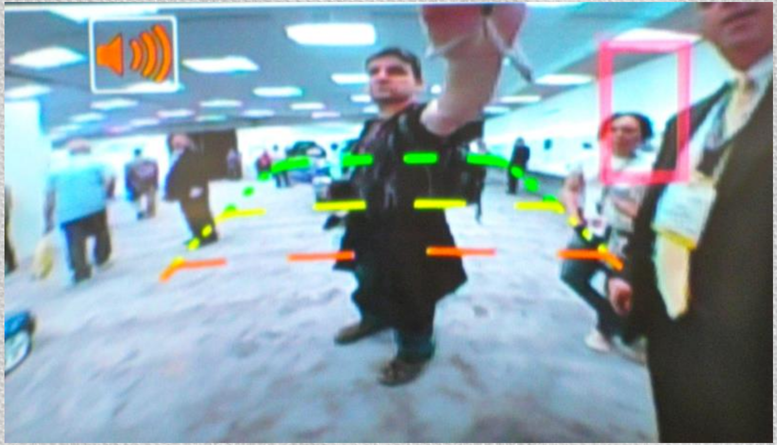
- Traditional automotive camera systems can detect pedestrians, cars and some other objects.
- Trade off between distance sensitivity and object sensitivity.



- Current approaches will take too much computational resources to attain both.

Why Detect Ground?

Detection Technology	Method	Target (Results)
Sonar sensor	Sound waves	Accurate distance. Does not know what the object is.
Object recognition	Computer vision	Pedestrians, cars, other "trained" objects. Poor distance accuracy.
Movement detection	Moving object against background	Anything that moves. Does not detect otherwise. Poorer distance accuracy.



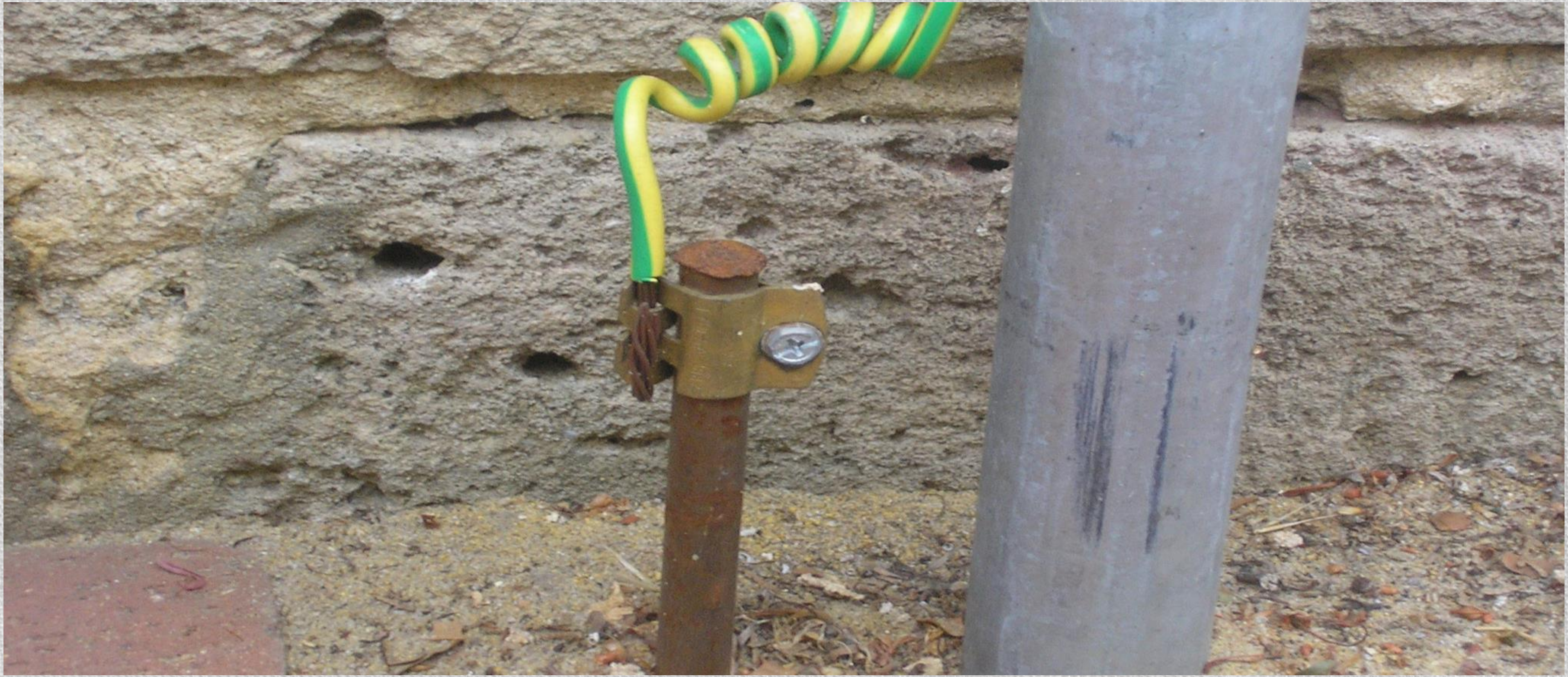
If we know the ground, we can detect both objects and their distances!

Our Project Goals

- Detect ground areas in backup camera images
  - Recognize various types of ground patterns
  - “Un-recognize” various non-ground objects and patterns

- Performance targeted to be near real-time
  - At or above 10fps will be considered acceptable
  - Less than 500 ms delay

What Does Ground Look Like?



Are These Ground?

Our Definition of Ground

- Fairly Smooth (Drivable)
- Even textured
- Discernable from the surrounding
- May have irregular but common imperfections

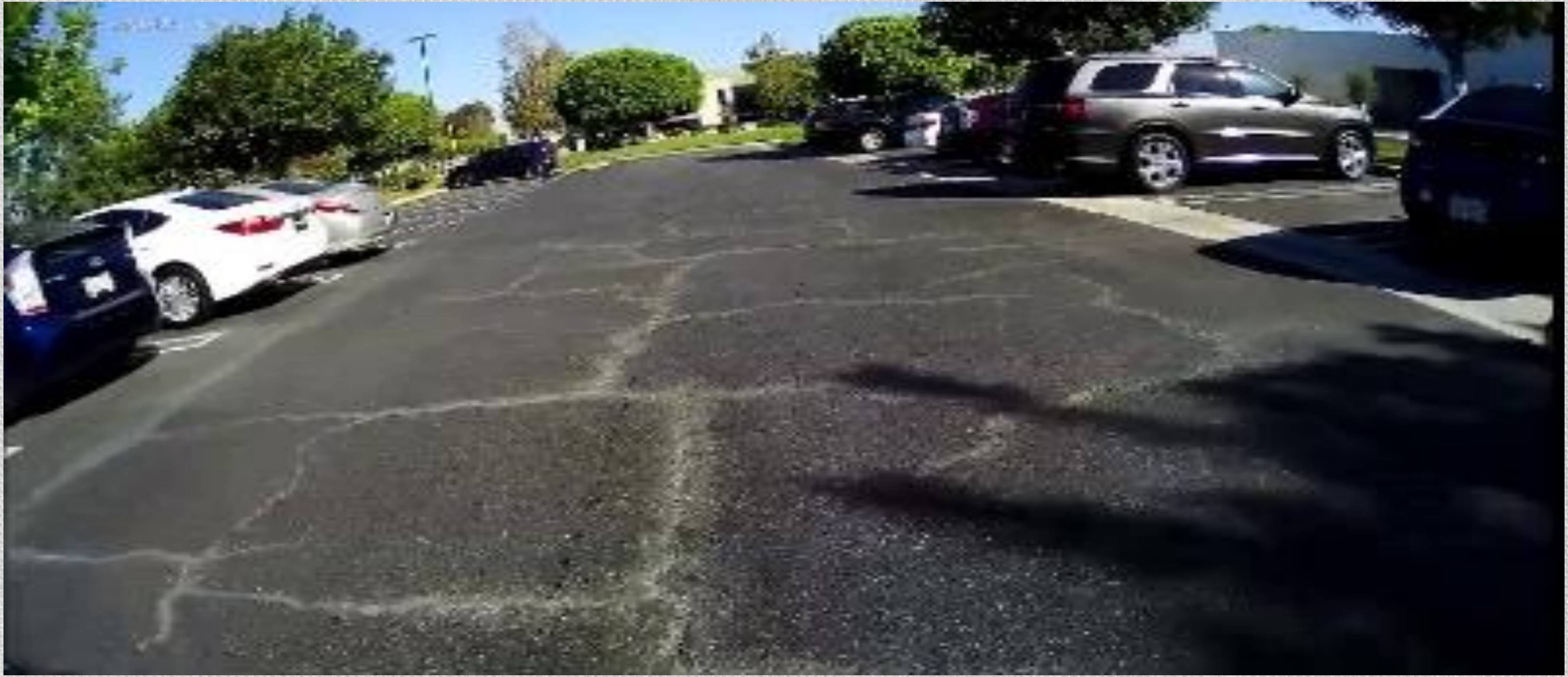
Why MATLAB & Coder

- Alternative choices
  - Native C/C++ development
  - OpenCV, OpenVX, PCL
  - Other open platform alternatives

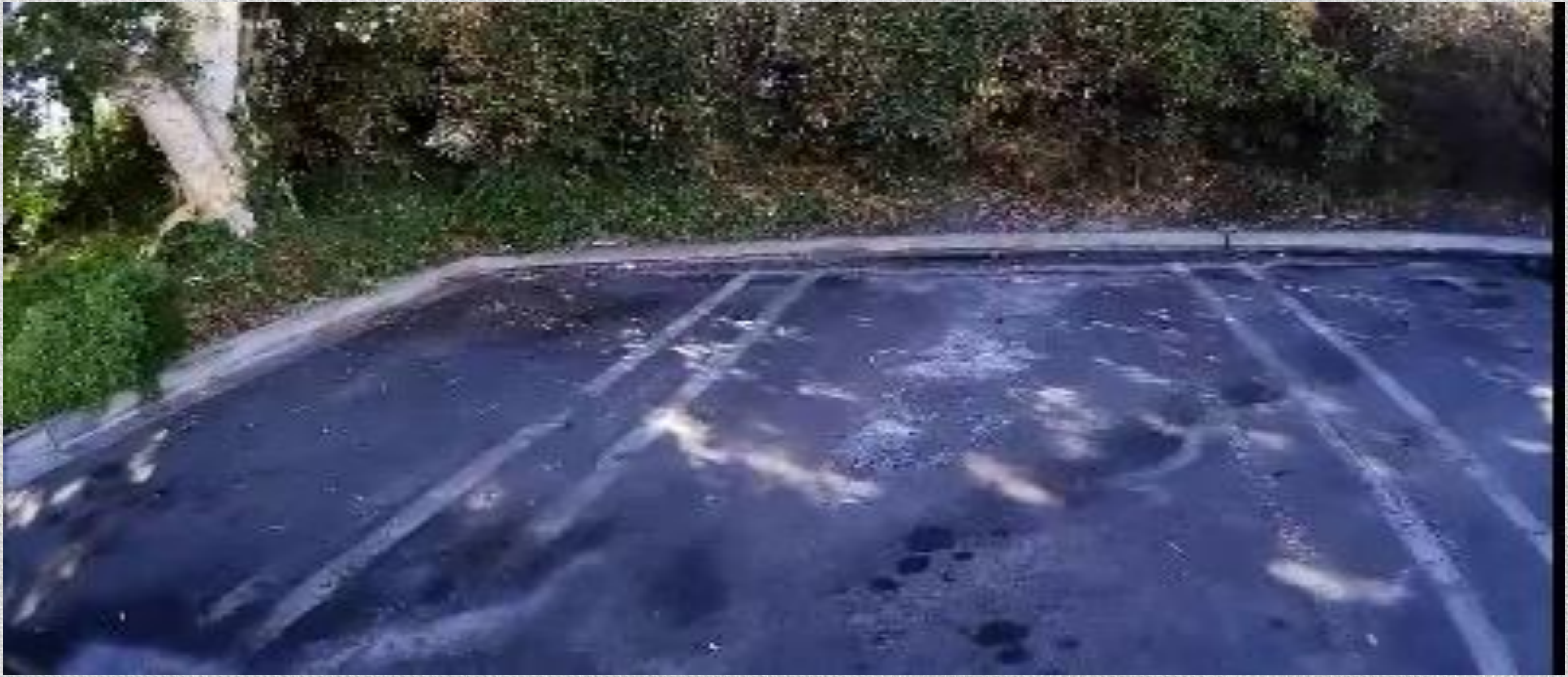
Why MATLAB & Coder

- Pros
  - + Fast development speed
  - + Familiarity of engineers with the toolboxes
  - + Quick turn around time for iterative development
  - + Great support from the professional team
  - + Direct conversion from MATLAB code to C/C++ code
  - + Seamless integration with C/C++ code (SIL)
  - + A rich collection of libraries and published material
- Cons
  - - Language not available in embedded environment
  - - Lack of automated build process for building target code
  - - Slower performance compared to native development

Sample Images – Imperfections

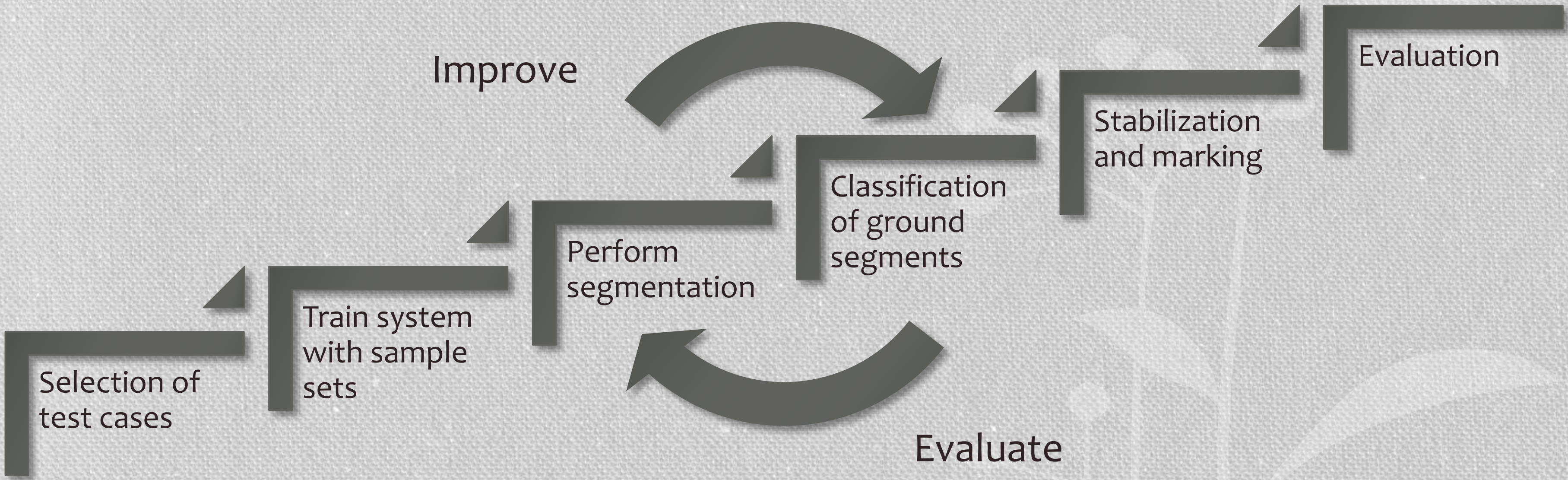


Sample Images – Light interference



Sample Images - Pedestrian

Development Process

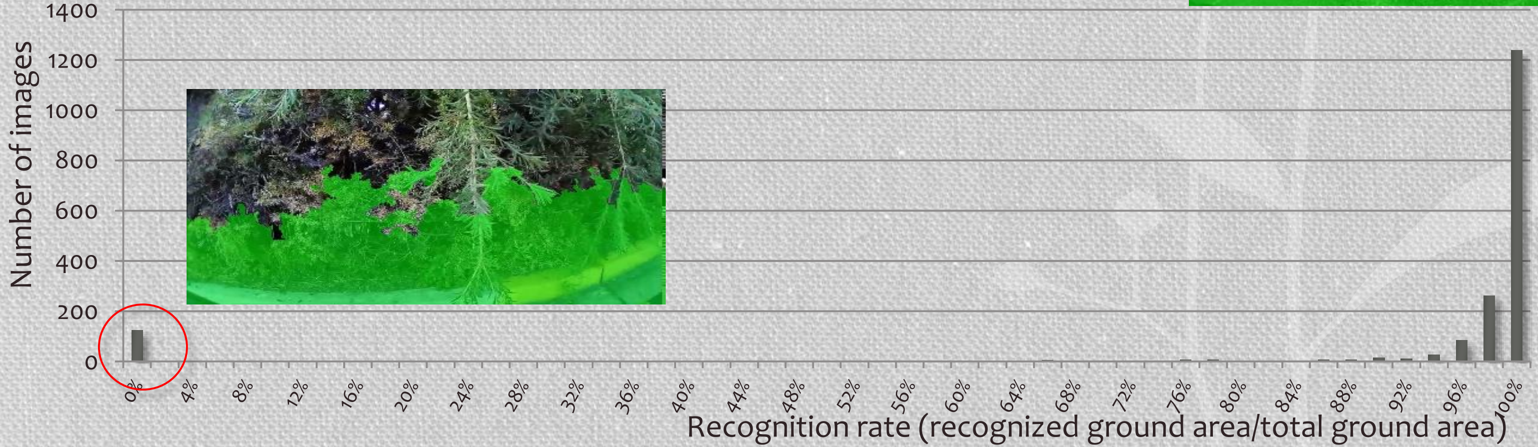

Improve
Selection of test cases
Train system with sample sets
Perform segmentation
Classification of ground segments
Stabilization and marking
Evaluation
Evaluate

Development Process – Test Cases

- Ten different test cases covering:
  - Parking lots
  - Parking roads
  - Ramps
  - Gates
  - Bushes
  - Shadows
  - Cars
  - Pedestrians

Results - Accuracy

- Achieved high recognition rates in our test cases.
  - Median recognition rate on ten test cases: 99%
  - Outliers at the low end due to “no ground” in view
- Slightly high false positive with a 23% median.
  - Non-ground being recognized as ground



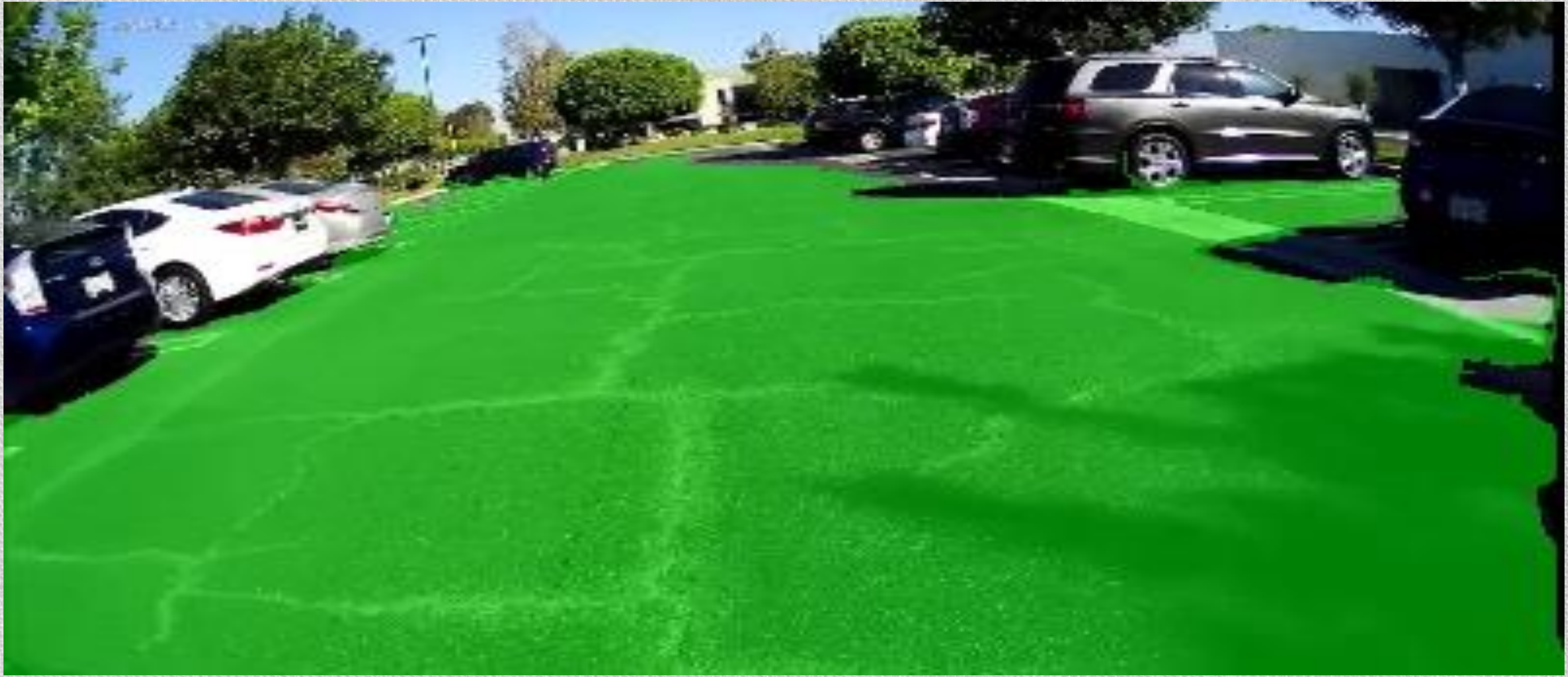
Results - Performance

- Near real-time performance with ~10fps in SIL mode for "pure" algorithm in Coder generated C.
- Achieved the delay target of ~500ms.
- Performance significantly dropped to ~5fps after removing all MATLAB dependency (e.g. data type dependency).
- Slowdown in converted C is attributed to:
  - Extra data type conversions or castings between functions
  - Less efficient memory management (compared to MATLAB)
  - Loss of multi-threading model from MATLAB
  - Loss of highly optimized MATLAB library code
  - Some Coder generated C-functions appear less efficient (than their MATLAB implementations)

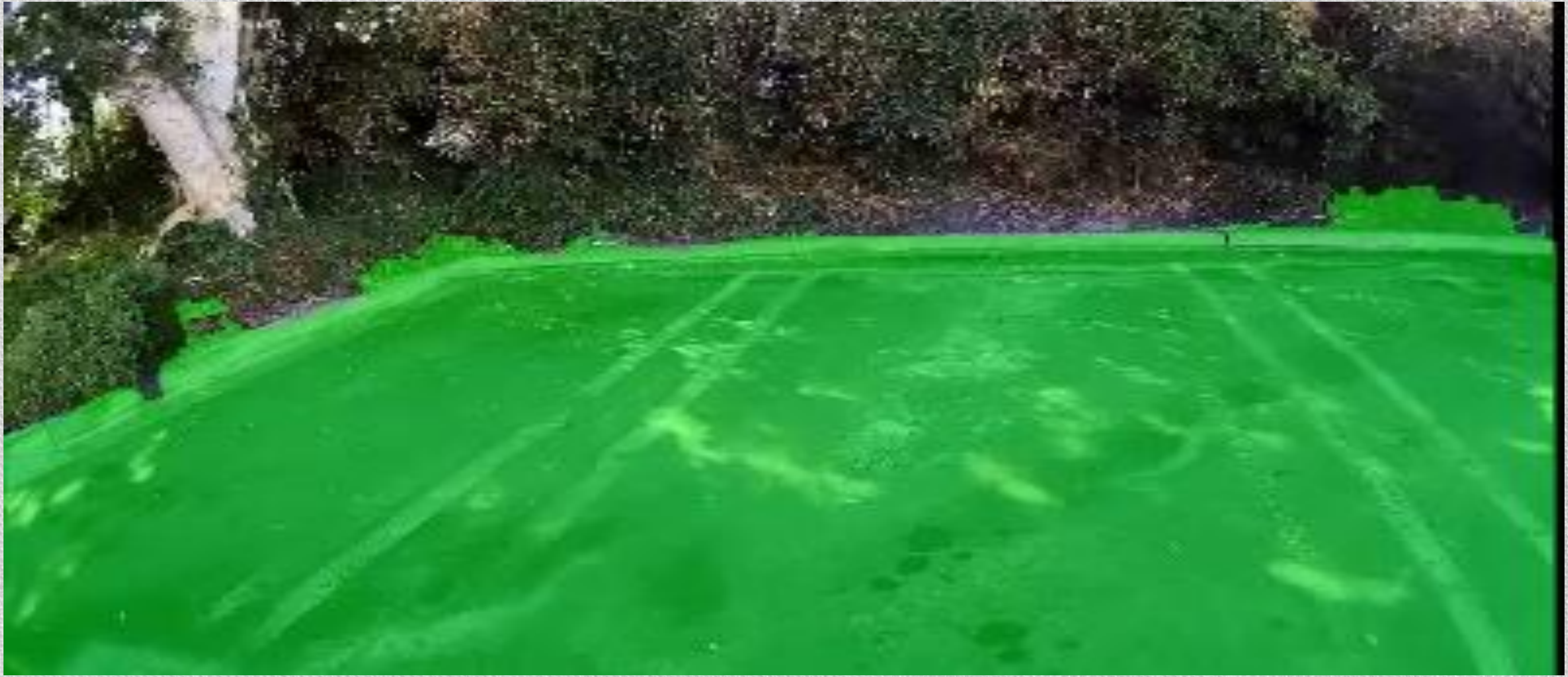
Sample Images

- Sample videos and images will be shown here..

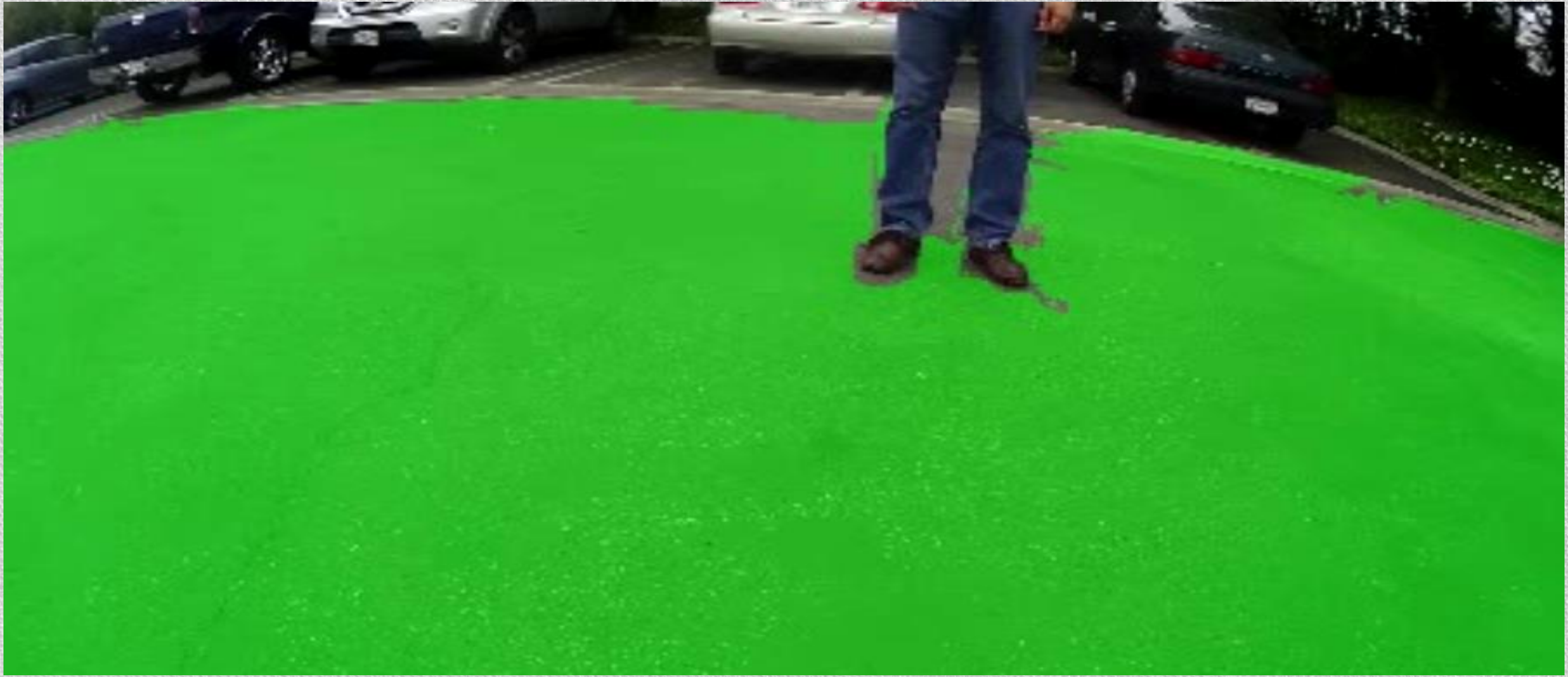
Results – Imperfections of ground



Results – Light interference



Results – Pedestrian



Sample Video

Conclusion

- MATLAB & Coder allowed us to focus on algorithm design
- Quick turn around time was ideal for experimentation
- Challenges in finding replacement or equivalent code in C for functions that cannot be converted by Coder
- Overall, MATLAB & Coder is a powerful tool for quick prototyping

- For future improvements:
  - Our existing MATLAB code could be refactored into modules for more flexible C-code generation

Acknowledgements

MathWorks Team

- Arvind Jayaraman
- Amit Deshpande
- Siva Nadarajah
- Pete Tsinzo

Alpine Team

- Priyen Shah
- Jun Cheng
- Dhruv Monga
- Kathy Li
- Joseph Chen

Thank you!