Hierarchical Navigation Architecture and Robotic Arm Controller for a Sample Return Rover

Velin Dimitrov, Mathew DeDonato, Adam Panzica, Samir Zutshi,
Mitchell Wills, and Taşkın Padır, *Member, IEEE*
Robotics Engineering Program, Worcester Polytechnic Institute
rover@wpi.edu

***Abstract*—This work presents a hierarchical navigation architecture and cascade classifier for sample search and identification on a space exploration rover. A three tier navigation architecture and inverse Jacobian based robot arm controller are presented. The algorithms are implemented on AERO, the Autonomous Exploration Rover, participating in the NASA Sample Return Robot Centennial Challenge in 2013 and initial results are demonstrated.**

***Index Terms*—sample return rover, driving with tentacles, histogram of gradients, inverse jacobian**

I. Introduction

The insatiable human curiosity and desire to explore proved Earth is neither the sole planet in the solar system nor the center of the universe. Technologies developed over the last 40 years have significantly enhanced humanity's ability to explore extraterrestrial bodies with teleoperated robots returning stunning images and scientific data of distant planets. In this work we present AERO, the Autonomous Exploration Rover, to advance the next evolution in space exploration. The next generation of exploration robots need to go where teleoperated robots cannot bring us.

AERO, shown in Figure 1, is comprised of a differential-drive four-wheeled mobility platform and 6-DOF manipulator designed to participate in the NASA Sample Return Robot Centennial Challenge. The task is to navigate a large outdoor area, find and locate various samples, and return them to the starting platform. Samples are defined in three broad categories: easy, medium, and hard. The easy samples are fully defined in terms of physical characteristics, the medium samples are defined in broad terms about general size, color, or texture, and the hard samples are vaguely defined, engraved with a small unique marking.

Fusing a combination of data from a fixed, forward-facing stereo vision system, LIDAR, and IMU, AERO implements a simultaneous localization and mapping (SLAM) algorithm to mark what areas are searched and return to the starting platform at the end of the competition. A second panning stereo vision system on a mast is used to locate and identify samples using object classifier and texture-based algorithms.

This work presents a navigation system and vision system that allow the rover to avoid obstacles, navigate towards the samples, recognize the samples, and retrieve them. At a conceptual level, the system is designed with hierarchical control layers. The supervisor, global planner, and local planner layers handle the state of the robot, current path to the target, and avoiding obstacles respectively.

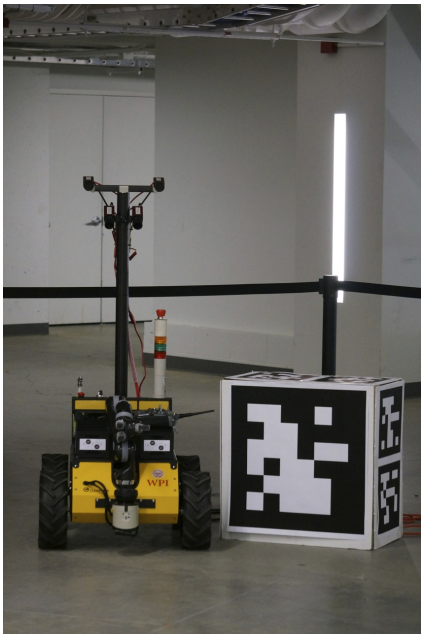

Fig. 1. AERO along with its VR tag homing beacon.

In order to locate and identify samples, AERO utilizes stereo vision object recognition, localization, and grasping algorithms to control the manipulator and retrieve the samples. Our algorithm first extracts the location information of the object of interest using disparity maps for location in 3D space. An object recognition algorithm determines the type of object and its orientation to plan a proper approach vector and grasping strategy.

AERO's misson can be split into three main subtasks: navigating and localizing within the large outdoor area, identifying and classifying samples, and retrieving the samples with a manipulator. The paper is organized in the following manner: Section II introduces the platform design of AERO, Section III outlines the previous work relevant to the navigation and vision algorithms, Section IV explains the architecture of the navigation system, Section V describes the local planner,

978-1-4799-0652-9/13 $31.00 © 2013 IEEE
DOI 10.1109/SMC.2013.761

Section VI explains the global planner, Section VII outlines the process for detecting and retrieving samples, Section VIII describes the controller for the manipulator, Section IX shows the preliminary results of our implementation, and Section X summarizes our work.

II. Platform Design

The system architecture is designed with the main sub tasks in mind. For example, AERO leaves the maximal amount of space on top of the robot for sample storage. The sensors are selected to comply with the competition rules, but also provide useful data to complete every subtask. A 6-DOF manipulator was selected to provide the most flexibility in sample handing, especially with the hard, undefined samples.

Inside AERO, a Roboteq MDC2250 dual output 60A motor controller implements closed loop velocity control on the primary drive motors. The control loop is closed using standard quadrature output optical encoders. The primary battery pack is also inside the robot towards the front consisting of sixteen 40Ah CALB LiFePO4 cells in an 8s2p configuration to provide 25.6V, 80Ah nominally. LiFePO4 cells were selected because of their good compromise between energy density, safety, and charge cycles. Two Manzanita Micro MK3x8 battery management systems ensure the safety of the lithium battery pack. Towards the front on a server motherboard, dual 8-core Intel Xeon processors provide the main computing hardware complemented by a NVIDIA Tesla K20 GPGPU co-processor. The Tesla GPGPU excels at image processing because of its highly parallel nature and significantly increases the vision processing capabilities of AERO.

A. Navigation Sensors

The main sensors AERO uses to navigate are the LIDAR, IMU, mast-mounted stereo vision system, and wheel encoders. GPS and other satellite based navigation aids are not used because they are not compatible with challenge rules. We selected a LMS151 LIDAR from SICK because of its 50 meter maximum range and excellent outdoor performance. The LIDAR directly feeds the SLAM algorithm by very accurately providing ranging data to trees and man-made features in the environment. A KVH 1750 fiber optic ring gyro IMU provides accelerations and angular velocities to enable AERO to dead-reckon when no good LIDAR features are available. A fiber optic ring gyro was selected because of its excellent stability and very low drift rates, providing accurate dead-reckoning for extended times without absolute positioning information from the LIDAR. The mast-mounted cameras periodically pan and extract trees from the scene to help localize the robot as well. Finally, wheel odometry from the motor encoders is fused with all the available data in an extended Kalman Filter (EKF) to localize the robot beter than any one sensor can by itself.

B. Sample Detection and Classification

Sample detection and classification is entirely implemented by the computer vision system. The top mast cameras identify anomalies in the grass that could potentially be samples and mark them on a probabilistic map on the robot. The robot inspects each potential sample from a close distance using the fixed, front-mounted stereo vision system. The easy and medium samples are identified and classified using a Linear Binary Pattern (LBP) classifier. Because the features of the easy and medium samples are known ahead of time, the robot is preloaded with a training set of data helping it identify these samples. The hard samples are identified by their generally different appearance in the environment. The metallic hard samples are extracted from the grass background using simple normalized RGB color filtering. In addition, the fixed forward facing vision system extracts the location, major axis, and bounding box of each samples in order to assist in planning a suitable approach vector for the manipulator.

C. Manipulation

A Kinova Jaco 6-DOF manipulator was selected to collect samples. It is a commercially available system that provides AERO with the needed flexibility to pick up samples of different sizes and place them on different locations on its top plate. The manipulator has a three-finger underactuated and compliant gripper with individual control over all fingers. The manipulator utilizes brushless DC motors with Harmonic drives resulting in low power consumption while still providing up to 1kg payload.

III. Previous Work

Our navigation algorithms are all based on the principle of driving with tentacles. Dillmann et. al. [1] present a method for extending the driving with tentacles algorithm [2]. They propose a system that allows for the control of an autonomous off-road vehicle that is more robust than the original work due to its ability to incorporate additional sensor data into the selection process. However, the algorithm only adds the ability to add additional obstacles and in a generally binary representation of existing. It did not adapt the underlying principle, and thus does not use the additional information to its fullest extent.

Quinlan et. al. [3] present a the concept of a real-time deformable global path in their work. They present a system where an initial global path is calculated, and as new sensor data is incorporated into the model the path is deformed locally to remain collision free. However, the computations are relatively expensive locally. In addition, the global path is never re-planned. If new sensor data has produced the possibility of a new global path that is more optimal, but beyond the reach of a local deformation, it will not be explored. Finally, collisions are only taken into consideration when deforming the path.

Mataric et. al. [4] present the concept of integrating global-tasks into behavior-based robots. In their method, a rough local path is planned, and behaviors are biased such that the robot will tend to follow the global path. In their work however, the behavior set is limited to a small series of hard-coded actions such as 'turn-left', 'go backwards', etc. The system also takes up all of the computational resources on the robot, but this work is fairly dated and may have been strongly

affected by technology limitations. Finally, there is no concept of exploring, the focus is solely on following a collision-free path between global points making unsuitable for space exploration.

The Haar algorithm in [5] and histogram of gradients (HOG) are two of algorithms already implemented in the OpenCV library for object detection. These are well known algorithms, and have been modified in various ways for many tasks. For the generation of disparity maps from stereo cameras, Doppelmann in [6] explains the important parameters such as SAD window size and demonstrates how they should be set. In our work, we use the standard OpenCV implemenations of these algorithms leveraging the performance optimizations of OpenCV.

IV. Navigation Overview

The implementation of driving with tentacles involves the generation of a set of arcs that emanate from the front of the robot. The navigation algorithm uses the following terminology.

Tentacle: A circular arc in space extending outwards from the front of the robot. Its properties are determined by the speed of the robot and the its position in a speed set.

Speed Set: A set of tentacles that correspond to a particular linear velocity.

Pseudo-Subsumption: A heuristic method to implement subsumption levels of in a behavior-based system, making them soft constraints instead of hard constraints.

Object of Interest (OoI): Objects on the field being explored which have been identified as samples.

A. Hierarchical Control

The AERO navigation system at its core uses the concept of hierarchical control which can be divided into three levels (Fig.2).

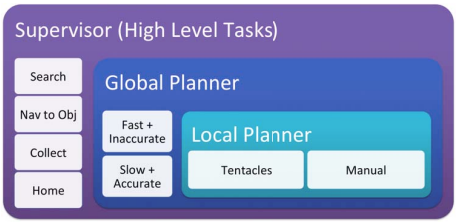

Fig. 2. The three levels of the hierarchical control. The supervisor coordinates the mission-level tasks. The global planner plans beyond the sensor horizon on the global map. The local planner plans on a local map representing a snapshot of the current environment around the robot.

The supervisor is the top layer of control implemented by a finite state machine and is responsible for mission level task planning and coordinating robot state. Examples of mission-level tasks include searching the field for objects of interest, navigating to a specific OoI, and collecting an OoI once it has been identified as a sample. It manages feedback from the OoI classification, sample collector control, as well as the navigation system. For the purposes of this work, we are only concerned with current mission task output of the supervisor.

TABLE I
Navigation System Failure Modes

Failure	Effect Up Chain	Effect Down Chain
Supervisor	NA	Global planner receives no new mission-tasks
Global Planner	Robot does not complete mission-task	Local planner receives no new goal points
Local Planner	Robot stops moving	NA

The global planner is the layer of control below the supervisor (Section VI). It is responsible for planning in the global scale and implementing the mission-task specified by the supervisor. The lowest layer of control is the local planner (Section V) which uses pseudo-subsumption. It is responsible for sending velocity commands to motor controllers and for dynamic obstacle avoidance.

B. Fault Tolerance

The hierarchical control allows the navigation system to have a high degree of fault tolerance. Systems further down in the chain can operate independently of systems higher up the chain. For example, if the supervisor fails, the global planner will simply continue attempting to complete its last mission-task. If the global planner fails, the local planner will attempt to reach the last goal it received following its standard behaviors, and then stop. Likewise, failures at the lower levels of the control hierarchy always leave the robot in a controlled, defined state. The failure modes are summarized in Table I.

V. Local Planner

The local planner implements the low level platform drive control and dynamic obstacle avoidance. As an input, it receives a goal in the global frame which it transforms into the local frame, and the current sensor data. It produces as its output the set of control velocities, $\begin{bmatrix} v & \omega \end{bmatrix}$.

A. Driving with Tentacles

Driving with tentacles is an efficient path-planning method [2] which utilizes a set of arcs (tentacles) as potential paths to follow for the robot. Each tentacle is defined by its radius and is grouped into speed sets. The tentacles in faster speed sets are grouped more closely together with smaller radii but a longer arc-length, and those in slower speed sets further apart with larger radii but smaller arc-length (Fig. 3). In [2], tentacle selection is performed by determining which tentacle is the longest before running into an obstacle given a radius of safety for the robot.

B. Extended Driving with Tentacles Algorithm

In order to improve the algorithm in [2], AERO implements a modified version which takes into consideration a number of behaviors suitable for a robot performing a search-and-return mission. The behavior-based extension of the algorithm is inspired by [7]. The modification combines the advantages of the tentacles algorithm, computational speed and simplicity and easy deterministic mapping to control outputs, with added

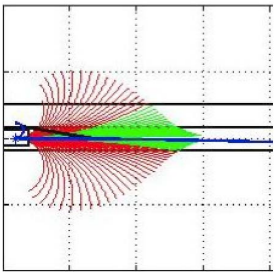

Fig. 3. A visualization of the tentacles in a speed set. As the robot's velocity increases, the tentacles shift closer to the center line and become longer.

behaviors aligned with the rover's mission. The behaviors include:

- Move towards some predetermined goal destination
- Move towards unexplored terrain
- Move away from previously explored terrain
- Move away from 'difficult' terrain

The desired end behavior will select the longest tentacle which moves the robot closest to the goal (if any) while weighing the passing through the least amount of previously explored terrain, the least amount of difficult terrain, and exploring the most new area. To accomplish this, the local planner is provided an occupancy grid generated by the global planner. For each new occupancy grid, the local planner iterates across each point on each tentacle in a speed set and examines the corresponding point in the occupancy grid's point trait. There are two values that are updated at each point $\mathbf{p}_{[n]}$ on tentacle k based on the point trait in the occupancy grid: The distance along the tentacle that has been traversed as in equation 2, and the length modifier given by equation 3. In addition, if the goal point or an obstacle is intersected, the iteration across that tentacle is halted.

$$\Delta l[n] = \|\mathbf{p}[n], \mathbf{p}[n-1]\|_2 \tag{1}$$

$$ll[n] = \Delta l[n] + ll[n-1] \tag{2}$$

$$lm[n] = lm[n-1] - \Delta l[n] \cdot \begin{cases} DW & \text{if FREE\_HIGH\_COST} \\ TW & \text{if TRAVERSED} \\ -UW & \text{if UNKOWN} \\ 0 & \text{if FREE\_LOW\_COST} \end{cases} \tag{3}$$

where $DW, TW,$ and UW are weighting parameters for biasing the behavior by their respective point trait.

$$lg = -GoalWeight \cdot \begin{cases} 0 & \text{If no goal} \\ \|\mathbf{p}_{[n_{end}]}, \mathbf{p}_{goal}\|_2 & \text{If goal} \end{cases} \tag{4}$$

$$l_f[n_{end}] = ll[n_{end}] + lm[n_{end}] + lg[n_{end}] \tag{5}$$

Equation 5 presents that the final effective length of the tentacle is the summation of its actual length before reaching any obstacles (if any) and modifiers based on the desired behaviors. Therefore the longest, and thus 'best' tentacle will naturally become the one which best satisfies the behavioral criteria. The priority of the behaviors can be adjusted by modifying the *DiffWeight*, *TravWeight*, *UnknWeight*, and *GoalWeight* parameters.

It is important to note that the local planner does not actually attempt to drive the entire length of the selected tentacle. It produces a single set of control inputs to the platform. With frequent updates and properly tuned weighting parameters, the robot is able to follow a path towards the goal while avoiding obstacles. No guarantees at all are made on the optimality of this path, or that it will not become stuck in local minima. When used in tandem with a global planner, however, the majority of these pitfalls can be avoided.

In addition, as this is mainly a heuristic approach, instability can result when two tentacles on opposite sides of the robot's center line have similar fitness values that swap back and forth rapidly. The resulting effect on the robot will be oscillations or jitter. To help damp these oscillations, a tunable rate-change damping coefficient or low pass filter is applied on the resulting tentacle selection. For example, if the previously selected tentacle was at index 30 and the next tentacle is selected to be at index 70 (the equivalent tentacle on the opposite side of the centerline), with the rate limit is set to 5, the damper will instead select tentacle 35. If tentacle 70 is selected again, it will select 40, and so on.

VI. Global Planner

The global planner is responsible for planning in the global world frame and completing the mission-level task specified by the supervisor. It takes as its input the current mission-task, and a locally centered global map generated by other means such as a SLAM implementation which is outside the scope of this paper. It produces as its output a series of way-points in the global frame that it can provide to the local planner. Due to the fact that global planner is not responsible for actually generating control inputs to the platform and that the local planner can improve the paths created due to imperfect data, the global planner can run at a low update frequency, on the order of once every 30 seconds. This saves computational resources on the robot needed for mapping and vision algorithms.

A. Carrot Path

Due to the behavioral properties of the local planner, the global planner can leverage its way-point connecting abilities to both save computation resources and compensate for imperfect data on the global map. Instead of attempting to produce a continuous path, the global planner instead produces a series of loosely-connected way-points called a carrot path. The carrot path strategy allows the global planner to relax many constraints on the path planner algorithm it uses. Even if it produces paths that are invalid because they cross an obstacle boundary, the local planner will compensate. If the point turns out to be unreachable due to imperfect data when the path was created, it will be corrected at the next re-planning phase. This would be undesirable in a robot operating in an enclosed

environment with very good mapping data, but is actually beneficial in an exploring robot in poorly mapped spaces as it will generally cause it to explore more of the map. It also again saves computational resources as the planning doesn't have to happen on a very fine search space.

B. Task-based Planning Strategies

The global planner also varies its strategies based on the current mission-task provided by the supervisor. Currently, there are two planning strategies that it uses: AStarCarrot, which is a simple modification of the A* algorithm to make it to produce carrot paths, and RRTCarrot, which is a similarly simple extension of the Bi-Directional Rapidly-exploring Random Tree Connect algorithm described in [8] to produce carrot paths and allow it to gracefully time-out with a partial path. Both planners operate in a purely 2D space. Orientation is ignored due to the fact that the only use-case that orientation matters, collecting an OoI, is handled by a control system that supersedes the navigation system entirely.

AStarCarrot is used when the robot is attempting to navigate to a specific point, such as an OoI. Its path-costs are based on the same weighting values as the local planner, and its heuristic is the Euclidean distance to the goal point. Its search-space is discretized to approximately $\frac{1}{10}th$ the distance between points in the Carrot Path, to ensure that the output is at least reasonably sure to be reachable. The final path is then culled to only contain points spaced for the carrot path.

RRTCarrot is used when exploring the global map for OoI's. Normally RRT algorithms are avoided in 2D space because they produce very sub-optimal and guaranteed inconsistent between planning runs paths. However, when searching for OoI's this random behaviour is desirable because it encourages a higher rate of exploration. To enforce at least some level of consistency and reduce the likelihood of the planner sending the robot to one end of the field in one plan and then the other end on the next plan due to random chance, a rough 'search pattern' of very distantly spaced way-points are connected in series using the RRTCarrot planner. Similarly to AStarCarrot, the 'step-size' when connecting nodes in the RRTCarrot is set to be approximately $\frac{1}{10}th$ the distance between points in the carrot path, and the final output is culled.

VII. Sample Detection and Retrieval

To detect the samples and classify them, we implement the standard OpenCV version of HOG algorithm. Fig. 4 details the flow of information in the vision system, and how the sample location is passed to the manipulator controllers. Once an object is detected, its 3D location information is extracted from the disparity maps generated by the stereo cameras. The manipulator controller then moves the arm to the desired position using a inverse Jacobian velocity controller that takes the current arm position and desired position as inputs.

In order to train the HOG, training images of each sample need to be provided to train the classifier. Initially, a cloth-lined and lit table-top stage was used to generate the images. This allowed the background to be easily removed from the image,

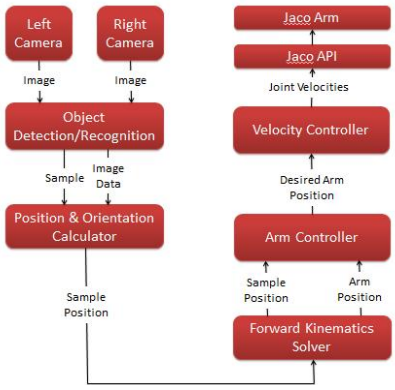

Fig. 4. A flow diagram showing the components of the vision system for detecting samples. The raw image streams are inputs, and after the samples is identified in the frame, it is passed to the stages of the manipulator controller to be picked up.

and leave just the sample in the image. After extensive testing and training sessions though, this method did not provide good detection rates.

In order to test the effect of the background on the training images, a new set of images was generated with a white noise background behind the sample. The detection rate using this set of images increased significantly. We presume the detection rate increases because the white noise background averages out and the classifier ignores it, unlike a plain background.

VIII. Arm Controller

The arm controller node determines the type of grip to use based off of the object classification, and its orientation. The arm controller calculates a rough path and approaches the object based on the best gripping strategy for the particular object. The velocity controller node receives a desired point from the arm controller, and converts it into a set of joint velocities required to get to the point. The first part of this node is the position control. When the desired position of the arm is received, the node first calculates the current position of the arm by feeding the joint angles, through the forward kinematics. Based off of the two points the linear and rotational errors are calculated. These errors are then each fed through separate PD controllers. The output of the controllers are then fed through a gain in order to convert the output into cartesian velocity. The second part of this controller uses the arm Jacobian calculated using the arm kinematics.

IX. Results

The navigation and vision algorithms described above were implemented with Robot Operating System (ROS) and OpenCV in Ubuntu Linux 12.04 on AERO. The software system was designed with flexibility and ease of implementation in mind. ROS provides a set of libraries and development tools on top of Linux tailored to robotics development that make it easy to develop software in multiple language across multiple systems. The ROS architecture involves creating a number of nodes that communicate using an inter-process message

passing system. In addition, there are a large collection of software tools and libraries that implement a number of algorithms and common tasks in robotics. Data collection for testing and visualization of data was made especially easy with these tools.

The test environment consisted of a series of impassible static obstacles, and one randomly moving dynamic obstacle near the center of the path from the start to the goal position. An example of the path taken by the robot can be seen in Fig. 5. It demonstrates the robot successfully dodging the dynamic obstacle while still moving towards the goal. A snapshot visualization of the data of this event can be seen in Fig. 6.

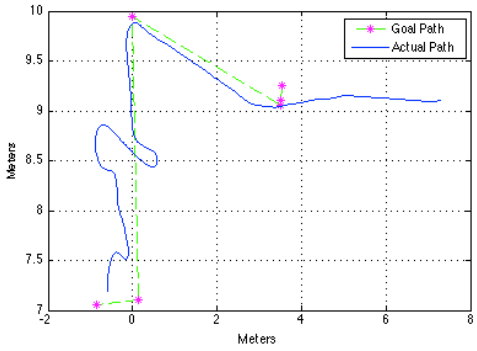

Fig. 5. A plot of the position of the robot in the global frame which includes a dynamic obstacle (a person moving in front of the robot) at location $x \approx 0.0, y \approx 8.75$. The blue line is the path the robot took. The green dashed line is the carrot path. And the purple stars are points where the carrot path was re-planned

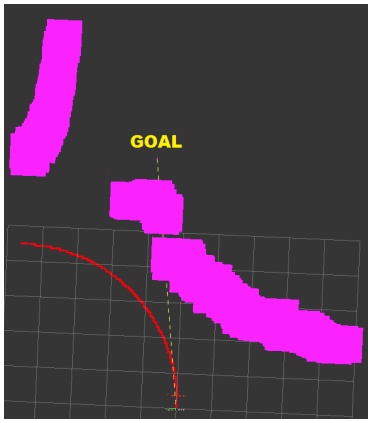

Fig. 6. Visualization of the robot dodging an obstacle.

Fig. 7 shows the output of the vision processing algorithms. The samples are detected, and their positions with respect to the arm are calculated. The arm controller then moves the manipulator into position to retrieve the samples. During benchtop testing, the sample was successfully retrieved multiple times.

X. Conclusion

This work presents the navigation and vision system architecture for a sample search and return planetary rover.

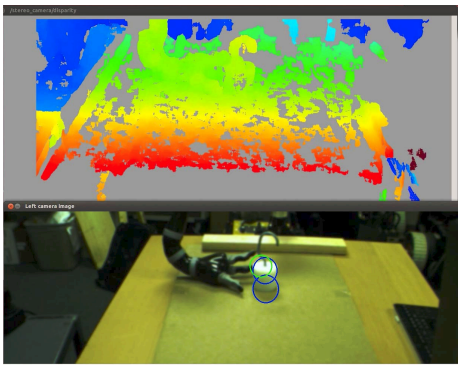

Fig. 7. Benchtop test showing the disparity map generated by the stereo cameras, and the two samples detected by the classifier. The sample positions are passed to the arm controller which moves the manipulator to retrieve the samples.

We implement a modified driving with tentacles algorithm with three hierarchical navigation layers. The layered approach provides advantages in its implementation including increased reliability over a single layer approach and control of its behavior using the proposed weighting scheme. The vision system leverages existing implementations of algorithms for object detection with training images, providing the positions of the samples in the work space. The arm controller uses that information to retrieve the samples.

Further testing over the next year will be conducted in varied outdoor environments, including in locations with rough terrain similar to extraterrestrial environments.

Acknowledgment

The authors would like to thank those who supported this work including AGCO, KVH, Clearpath Robotics, Harmonic Drive LLC, NVIDIA, Dragon Innovation, Gigavac, EKWB, Lincoln Tool, and Advanced Circuits.

References

[1] M. Manz, M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Fusing lidar and vision for autonomous dirt road following," in *Autonome Mobile Systeme 2009*, ser. Informatik aktuell, R. Dillmann, J. Beyerer, C. Stiller, J. Zllner, and T. Gindele, Eds. Springer Berlin Heidelberg, 2009, pp. 17–24.

[2] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008. [Online]. Available: http://dx.doi.org/10.1002/rob.20256

[3] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, may 1993, pp. 802 –807 vol.2.

[4] M. Mataric, "Integration of representation into goal-driven behavior-based robots," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 3, pp. 304 –312, jun 1992.

[5] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 1, 2002, pp. I–900–I–903 vol.1.

[6] S. Droppelmann, "Stereo vision using the opencv library," June 2010.

[7] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14 – 23, mar 1986.

[8] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, 2000, pp. 995–1001 vol.2.