# Performance Evaluation of the Primitive Mobility Execution of an Automated Guided Vehicle within the Mobility Open Architecture Simulation and Tools Environment

Will Shackleford
National Institute of Standards and Technology(NIST)
100 Bureau Drive, Stop 8230, Gaithersburg, MD 20899
(301) 975-4286, shackle@nist.gov

## ABSTRACT

This paper describes the role of the Primitive Mobility (PrimMob) module within the Mobility Open Architecture Simulation and Tools(MOAST) environment. Descriptions are given of several alternative implementations of the module motivated by a desire to make MOAST more usable for an industrial Automated Guided Vehicle(AGV). A series of performance metrics is described as a series of tests that allow those performance metrics to be determined. Tools added to MOAST in 2009-2010 that greatly ease recording and visualization of these performance metrics are also described.

## Keywords

MOAST, Data-Collection, Automated Guided Vehicles(AGV)

## 1. INTRODUCTION

The Mobility Open Architecture Simulation and Tools(MOAST) system[1],[2] is a hierarchical architecture that is an implementation of National Institute of Standards and Technology(NIST) [3] Real-Time Control System(RCS) [4] reference model architecture[5]. It is intended as a general purpose research tool with applications ranging from low-level robotic control to behavior generation for groups of heterogeneous robots. In addition to research applications, MOAST has been used as the control system for teams competing in the RoboCup rescue virtual robot competition (RoboCup)[6] as well as the IEEE Virtual Manufacturing Automation Competition (VMAC) [7], [8]. MOAST was also previously used in evaluation of range imaging sensors for the ANSI/ITSDF B56.5 safety standard[9].

Automated or Automatic Guided Vehicles (AGVs) are mobile robots typically used in an industrial setting to move materials around a factory or warehouse[11]. There are a wide variety of AGVs. They vary depending on the load to carry, the type of infrastructure installed to allow them to determine position (buried wires, laser reflectors), and whether the AGV will also be manually driven, the steering mechanism[12] etc. They differ from autonomous vehicles used in research and military applications in that paths are preplanned off-board. Preference is given to having repeatable, reliable, predictable, and safe operation over the ability to search unknown terrain to find paths in unstructured environments.

As part of the Mobility and Manipulation Project [13], it was decided to adapt MOAST for use in an industrial scenario. A first attempt was made to utilize the entire autonomous navigation capabilities in MOAST. However, this turned out to be unsatisfactory for several reasons:



- Many hazards either to the vehicle or from the vehicle to personnel or equipment in the environment could only, be identified by trained safety experts and not by the vehicle's on board systems. These included overhanging obstacles, wire fences, glass dividers, negative obstacles and places where people or equipment might move in the future. ( Negative obstacles are obstacles with negative height such as holes in the ground. ) Adding additional sensors and more sophisticated sensor processing and world modeling might have alleviated the problem but it was beyond the budget and scope of this project.
- One approach for dealing with these undetectable obstacles was MOAST's support for virtual obstacles. This is a list of obstacle positions stored in a file a priori. However this was also insufficient: errors in the positioning system tended to move virtual obstacles too close to detected obstacles.
- The map used by the autonomous system was too coarse to allow the AGV to negotiate corners with very tight tolerances within the safety fence surrounding the robot. Adjusting the resolution of the map was possible, however the software development cost and time were deemed excessive.
- Since the autonomous navigation level constantly updates the path, it was impossible to tell whether path following problems were caused by unacceptable paths, or if there was an underlying problem in the lower levels.

For these reasons it was decided to focus on the Primitive Mobility level.

## 2. MOAST Primitive Mobility

For a full overview of MOAST see [14]. Here we are only interested in isolating and testing the Primitive Mobility(PrimMob) module as highlighted in Figure 1. The PrimMob level normally expects the higher level Autonomous Mobility(AMMob) level to combine sensor data and provide an obstacle free path to goals selected by the Vehicle level or above. The Servo Mob level handles the electronic hardware specific tasks including interfacing with the digital-to-analog converter (DAC) and tuning the output to match the amplifiers and motors to achieve the commanded

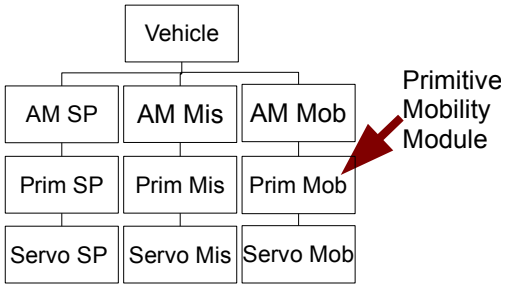


**Figure 1: Moast Hierarchy For a Single Vehicle, highlighting PrimMob.**

wheel speeds. The modules in the mission(Mis) column handle systems that do not affect mobility such as the conveyor on a unit loader. The modules in the sensory processing(SP) column handle analysis and collection of data from sensors.

While there are several housekeeping commands to respond to and other auxiliary inputs and outputs, we can simplify the analysis by considering only two inputs and one output:

- "primMobJACmd1" buffer, which will at anytime contain either a message of type
  - PrimMobJACmdMoveWaypoint -- a list of Cartesian waypoints implying line-segments between them form an obstacle-free path to the goal.
  - PrimMobJACmdMoveArcSegment – a list of a mix of line segments and circular arcs providing an obstacle-free path to the goal.
- "navDataExt1" buffer that contains a message of type NavDataExt that contains the vehicle's current measured position {x,y} and orientation yaw.
- "servoMobJACmd1" is written by PrimMob and read by ServoMob and contains a message of type ServoMobJACmdSkid since our AGV is a skid-steer or differential drive type. The AGV has two wheels on the left and two wheels on the right. It is steered by commanding a different speed to the right and left wheels. If the right wheels are commanded to rotate faster, it turns right. If the left wheels are commanded to rotate faster, it turns left. For sharp turns, it can turn in place by having one side reverse direction. ServoMobJACmdSkid has two variables(wLeft and wRight) which are the left and right commanded rotational wheel speeds in radians/second.

All of the implementations of PrimMob need to convert the translational forward velocity ($V_T$) in meters/second and rotational velocity ($V_R$) in radians/second to the wheel velocities, also in radians/second. This is done by using the constant separation distance between the wheels($WHEEL_{separation}$) and the constant radius of the wheels($WHEEL_{radius}$) as shown in Equation 1.

$$wLeft = \frac{1}{WHEEL_{RADIUS}} * (V_T + \frac{V_R * WHEEL_{SEPARATION}}{2})$$

$$wRight = \frac{1}{WHEEL_{RADIUS}} * (V_T - \frac{V_R * WHEEL_{SEPARATION}}{2})$$

**Equation 1: Compute Skid Steer Wheel Velocities from forward translational velocity and rotational velocity.**

The remaining algorithms describe how to compute the vehicle's translational and rotational speeds $V_T$ and $V_R$. These speeds are expressed in a Cartesian coordinate system independent from vehicle configurations, making these calculations easily adaptable to a variety of vehicle types. In the tests described here, they are computed periodically, nominally every 10 milliseconds.

## PrimMobJACmdMoveWaypoint

Of the two commands we will deal with, the simpler one is the PrimMobJACmdMoveWaypoint command that provides a list of up to one hundred waypoints. Each waypoint has four variables of interest.

{x,y} – Cartesian X and Y coordinates in meters of the end of the line segment.

neighborhood – The maximum allowed deviation from the line segment in meters.

speed – Maximum forward translational speed in meters/second allowed along line segment ending in this waypoint. A negative speed indicates that the vehicle should back up. In the rest of this paper speeds will be assumed to be positive. All algorithms here handle negative speeds by assuming the vehicle heading to be reversed and then treating the speed as positive.

There is also a z value that will be ignored since the vehicles are only capable of moving in 2 dimensions. An available roll-pitch-yaw (rpy) and quality variable were also not used in these tests.

## PrimMobJACmdMoveArcSegment

PrimMobJACmdMoveArcSegment is more general and contains a list of up to one hundred ArcSegments. Each ArcSegment contains either an arc or a line, distinguished by a flag. Arcs are specified additionally with the coordinates of the arc center. The radius of the arc must be computed by determining the distance between the end point and center point. The angle of the arc can be computed using the previous end point as the start of the arc. A normal is provided but ignored in these tests since in two dimensions only arcs around the z axis are possible. An annular tolerance is used with arcSegments instead of a neighborhood. The translational maximum velocity associated with each Waypoint or ArcSegment will be denoted as Vi for the $i^{th}$ Waypoint.

## Global Constraints

In addition to the constraints included in the path there are four global constraints set in the configuration file:

$V_{TMAX}$=Maximum Translational Velocity in m/s

$A_{TMAX}$=Maximum Translational Acceleration in $m/s^2$

$V_{RMAX}$=Maximum Rotational Velocity in deg./s

$A_{RMAX}$=Maximum Rotational Acceleration in $deg./s^2$

# 3. PrimMob Algorithms

## Proportional Error Algorithm: A0

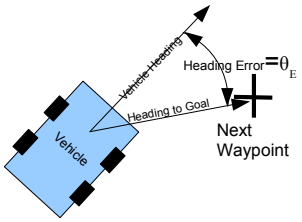


**Figure 2: Diagram showing how the Heading Error is calculated.**

Algorithm A0 sets both $V_T$ and $V_R$ based on the difference between the current heading and the heading along which the goal or next waypoint could be achieved ($\theta_E$), as shown in Figure 2. The algorithm can be tuned by setting two parameters: $\theta_{VCUTOFF}$, the angle at which translational velocity will be cut off, and $\theta_{WCUTOFF}$, the angle at which rotational velocity will be clipped to $V_{RMAX}$.

$$V_T = \begin{cases} V_i * (1 - \dfrac{|\theta_E|}{\theta_{VCUTOFF}}) & if\ |\theta_E| < \theta_{VCUTOFF} \\ 0 & if\ |\theta_E| > \theta_{VCUTOFF} \end{cases}$$

$$V_R = \begin{cases} -V_{RMAX} & if\ \theta_E < -\theta_{WCUTOFF} \\ \dfrac{V_{RMAX} * \theta_E}{\theta_{WCUTOFF}} & if\ |\theta_E| < \theta_{WCUTOFF} \\ +V_{RMAX} & if\ \theta_E > \theta_{WCUTOFF} \end{cases}$$

**Equation 2: Equations to compute translational and rotational velocities for algorithm A0 before acceleration constraints are applied.**

After Equation 2 is applied, the values may need to be clipped to enforce the acceleration constraints. In order to enforce the acceleration constraints, it is necessary to know the last translational velocity $V_{TL}$, the last rotational velocity $V_{RL}$, the cycle time $T_C$, and the distance to the end of the path D. $V_T$ and $V_R$ computed using Equation 2 will be denoted as $V_{Tin}$ and $V_{Rin}$. For convenience, we also introduce $V_{Tll}$, $V_{Tul}$ ,$V_{Rll}$, $V_{Rul}$ as the temporary upper and lower limits for translational and rotational velocities, respectively. $V_{Tdecel}$ is a temporary variable for the velocity required to decelerate to zero at the end. $V_{Tfinal}$ an $V_{Rfinal}$ are the final output translational and rotational velocities.

$$V_{Tdecel} = min(V_{Tin}, \sqrt{(2 * D * A_{TMAX})})$$

$$V_{Tul} = V_{TL} + A_{TMAX} * T_C$$

$$V_{Tll} = V_{TL} - A_{TMAX} * T_C$$

$$V_{Tfinal} = \begin{cases} V_{Tul} & V_{Tdecel} > V_{Tul} \\ V_{Tdecel} & if\ V_{Tll} < V_{Tdecel} < V_{Tul} \\ V_{Tll} & V_{Tdecel} < V_{Tll} \end{cases}$$

$$V_{Rul} = V_{RL} + A_{RMAX} * T_C$$

$$V_{Rll} = V_{RL} - A_{RMAX} * T_C$$

$$V_{Rfinal} = \begin{cases} V_{Rul} & V_{Rin} > V_{Rul} \\ V_{Rin} & if\ V_{Rll} < V_{Rin} < V_{Rul} \\ V_{Rll} & V_{Rin} < V_{Rll} \end{cases}$$

**Equation 3: Equations used to enforce Acceleration Constraints**

ArcSegments are handled by converting each ArcSegment into a series of waypoints each separated by the annular tolerance along the arc.

Finally, a rule must be defined for determining when to increment the path index. The path index is always set to zero when a new command is received and incremented when the distance to the current waypoint is less than the neighborhood associated with that waypoint. If the distance to the last waypoint is less than the last neighborhood, motion is stopped and a DONE flag is sent to the supervising module (AMMob) until the next command is received.

## Constant Arc Radius Algorithm: A1

One limitation of algorithm A0 is that when the $\theta_E$ peaks generally at the beginning of each segment, $V_T$ will be very small or possibly zero, which robs the AGV of momentum. To keep $V_T$ closer to its maximum we can delay the rotation and distribute it over the entire curve.

$$R = \begin{cases} \dfrac{Y_V}{2} & X_V \le 0 \\ \infty & Y_V = 0 \\ \dfrac{(X_V^2 + Y_V^2)}{2 * Y_V} & Y_V \ne 0 \wedge X_V > 0 \end{cases}$$

$$V_T = \begin{cases} V_i & V_i < |V_{RMAX} * R| \\ \dfrac{V_{RMAX}}{|R|} & V_i > |V_{RMAX} * R| \end{cases}$$

$$V_R = \begin{cases} V_{RMAX} & V_i > |V_{RMAX} * R| \wedge Y_V > 0 \\ \dfrac{V_i}{R} & V_i < |V_{RMAX} * R| \wedge Y_V \ne 0 \\ 0 & Y_V = 0 \\ -V_{RMAX} & V_i > |V_{RMAX} * R| \wedge Y_V < 0 \end{cases}$$

**Equation 4: Compute Translational and Rotational Velocities using algorithm A1**

With this algorithm each waypoint is first transformed into vehicle-relative coordinates by subtracting the currently sensed navigation position and rotating the coordinates based on the current yaw orientation. A special case is made for when $X_V$ is less than zero, i.e., when the next waypoint is behind the vehicle. It was decided to do a 180 degree turn followed by a straight line rather than an arc that might be huge compared to the distance to the waypoint. The radius of curvature(R) and the initial values for $V_T$ and $V_R$ are computed with Equation 4. Just as with algorithm A0, the acceleration constraints are enforced by applying Equation 3; however, as with all the following algorithms, $V_{Tin}$ and $V_{Rin}$ are first computed by projecting ahead along the path that the vehicle would follow if commanded $V_T$ and $V_R$ for time $T_C$. This leaves us with a new projected point and potentially a new next waypoint, so the primary algorithm (in this case Equation 4 ) is reapplied. If the new projected point has a lower $V_T$ than the current point then Equation 5 is applied, where $V_{Torig}$ and $V_{rorig}$ are simply the original values of $V_T$ and $V_R$. After each projection the new $V_{Tprojected}$ and $V_{Rprojected}$ are used to compute the next projected point until $D_{projected}$ is greater than $(V_T)^2/2*A_{TMAX}$.

$$D_{projected} = \sum_{j=0}^{j=i} T_C * V_{Tprojected}(j)$$

$$V_{TMAXATSTART} = \frac{V_{Tprojected} + \sqrt{V_{Tprojected}^2 + 8 * A_{TMAX} * D_{projected}}}{2}$$

$$scale = min(V_{TMAXATSTART}, V_{Torig}) / V_{Torig}$$

$$V_T = scale * V_{Torig}$$

$$V_R = scale * V_{Rorig}$$

**Equation 5: Use projected velocities to recompute current velocities.**

The path index to the next waypoint is incremented under the same condition as with A0 when the distance to the waypoint is less than the neighborhood; however, it also increments the waypoint index if the distance to the next line segment is less than the distance to the current line segment. This eliminates the need to spiral around a waypoint that was missed due to positioning noise or a very small neighborhood. ArcSegments do not need to be interpolated as with algorithm A0. Instead the end point of the arc is simply treated as another waypoint.

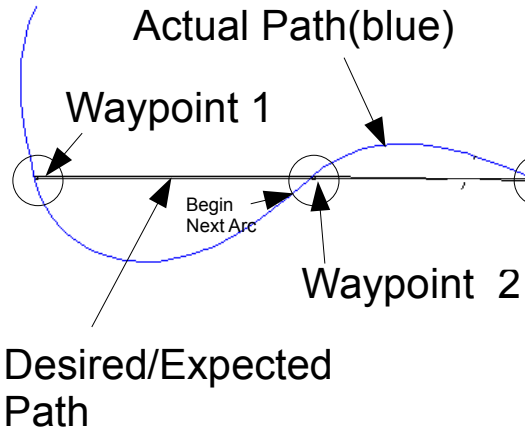


**Figure 3: Example Path produced by A1**

Algorithm A1 is good at maintaining a high translational velocity. However, the arcs may deviate greatly from the straight line segments between points. Additionally, oscillations can be produced as the vehicle overshoots the line after passing though each waypoint as in Figure 3. The oscillations decrease each time since the vehicle begins an arc to the next waypoint before reaching the current waypoint and therefore before overshooting. One solution would be to add additional intermediate waypoints to cause the oscillations to decrease more rapidly and thereby keep the vehicle closer to the intended path. Adding additional waypoints can also reduce the distance A0 deviates from the path although it is less critical since a similar effect can be achieved by reducing the cutoff angles ( $\theta_{VCUTOFF}$ and $\theta_{WCUTOFF}$ ).

## Fixed-distance ahead Algorithm : A2

While algorithm A1 is impractical for an AGV in the general case where one can not ensure closely spaced waypoints, it can be adapted by replacing the next given waypoint with an effective waypoint that is always a fixed chosen distance ahead ($L_{ahead}$) along the path as shown in Figure 4. To find the effective waypoint, the vehicle's current position is projected onto the closest segment between waypoints and then the length to each successive waypoint is added until the the sum of the distances exceeds $L_{ahead}$. The effective waypoint is chosen along the segment just before the distance was exceeded based on the remaining distance of $L_{ahead}$ minus the sum of the previous distances.

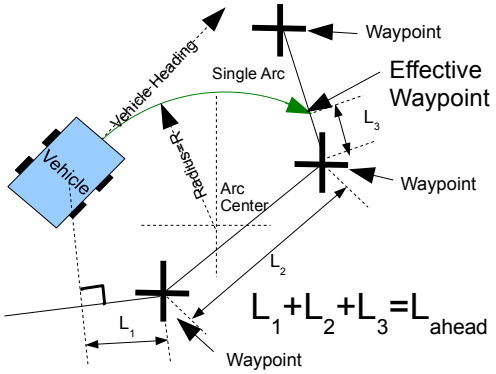


**Figure 4: Diagram showing selection of effective waypoint for Algorithm A2**

The points can be projected onto arc segments in a similar fashion. If $L_{ahead}$ is chosen smaller than the neighborhood or tolerance then the vehicle will generally stay within tolerance. If it is chosen larger the vehicle will begin to cut to the inside of corners.

## Finding the Longest Path Algorithm : A3

While A2 removes the large overshoots of A1 and never brings the vehicle to a complete halt like A0, it still concentrates most of the turning near the waypoints with a corresponding decrease in speed. A3 can be more aggressive than A2 while keeping the overshoot strictly limited by the neighborhood specified in the path. The goal is to find an effective waypoint with the largest possible $L_{ahead}$ at a given location that provides an arc which stays within the tolerances.

The algorithm requires repeated application of Equation 4. Instead of just using the next waypoint as A1 would do we check each waypoint starting with the current one to see if an arc as computed with Equation 4 would ever reach a point that was farther away from all segments than the neighborhood associated with that line segment would allow. If the arc does not exceed any neighborhood limit the next waypoint is tested. As soon as an arc to a waypoint fails this test, we test the midpoint between the last successful waypoint and the first failing waypoint. We continue to test the midpoint between the last successful point and the first failing point on the line segment between these points until the difference between the points is less than an arbitrarily-chosen minimum, here 5mm. After the curve is chosen, the same projections to compute speed to limit acceleration are performed that were done in A1 and A2. Arc segments are handled almost the same as waypoints, except when determining whether the arc is within tolerance, we check its distance from the arc segment.

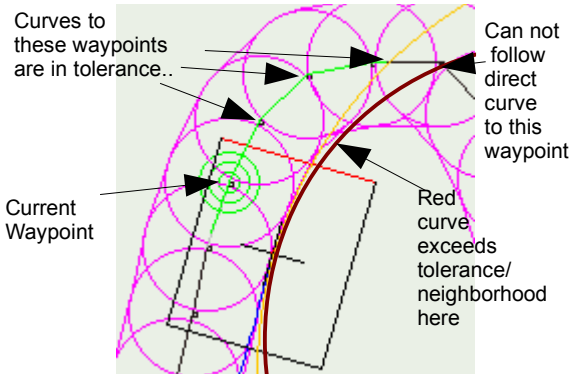


**Figure 5: Diagram showing selected curve in orange and a curve that was tested and rejected in dark-red. There is a purple circle with a radius equal to the neighborhood around each waypoint. The orange circle shows the algorithm's output command.**

One additional modification is needed to bring the vehicle back to the path if it is outside the neighborhood. The servo controller may not respond as commanded or the position system may suddenly reacquire an absolute position marker and discover the position is already outside of the tolerance. In this situation, A3 would be unable to find an arc since all arcs would be at least partially outside the tolerance. It might also be unable to find an arc while currently inside the allowed area, but already pointed so far away from the path that all circular arcs leave the allowed area. The solution chosen is to revert to algorithm A2 and simply choose an $L_{ahead}$ equal to the neighborhood associated with the next waypoint even if that results in a path farther from the commanded path than a tolerance would allow.

## Limited Neighborhood Algorithm : A4

A3 keeps speeds as high as possible while staying within tolerance, at least as long as the servo layer behaves perfectly. However, it tends to ride the edges: after a left turn it will end up on the right edge of the neighborhood, and after a right turn it will end up on the left edge of the neighborhood, only briefly passing through the center in either case. Equation 6 reduces the effective neighborhood (N) from the original neighborhood specified in the command ($N_{orig}$) whenever the arc length (L) is greater than some chosen minimum ($L_{min}$). The neighborhood is never allowed to reach zero but instead kept to a fraction ($N_{minscale}$) of the original neighborhood if the arc length exceeds a chosen maximum ($L_{max}$). All of our tests were done with the following settings:

$$L_{min} = 5*N_{orig}$$

$$L_{max} = 25*N_{orig}$$

$$N_{minscale} = 0.5$$

$$N = \begin{cases} N_{orig} & L < L_{min} \\ N_{orig} * (1 - \frac{(1 - N_{minscale}) * (L - L_{min})}{L_{max} - L_{min}}) & L_{min} < L < L_{max} \\ N_{orig} * N_{minscale} & L > L_{max} \end{cases}$$

**Equation 6: Limit Neighborhood to Bring Vehicle back to Center**

## 4. Metrics

For each test we will report the following metrics.

- $E_{max}$ = maximum error = maximum value of the distance from the commanded path minus the tolerance specified for that section of the path
- $T_{move}$ = move time = time between the command being sent and a corresponding status of DONE being received. In all tests the vehicle reached Table 1 neighborhood of the last waypoint and stopped motion before reporting DONE.

## 5. Tools

Two new tools were developed and added to MOAST to support these tests.

- moastLogRecorder – A program that runs in the background and records all data sent from any given set of MOAST Neutral Message Language (NML) buffers in NML Packed Message files[17].
- PathDraw – A graphical application that allows the user to draw, save and reload sets of waypoints and arc segments and send these as commands to either the AM or PRIM level. It also provides a real-time display of the vehicle's position and orientation as well as obstacles detected by the laser line scanner, virtual obsacles, and a trace of previous positions. It is also capable of reading the files recorded by the moastLogRecorder and computing the metrics.

## 6. Tests

Each test consists of a starting point and a set of waypoints or arc segments stored in a file that could be converted to a PrimMobJACmdMoveWaypoint or PrimMobJACmdMoveArcSegment command and sent to the primMobJACmd buffer. MOAST can be configured to use either the Player/Stage/Gazebo interfaces for simulation or USARsim (Unified System for Automation and Robot Simulation). Player is a network server for robot control[16]. Two simulators are built to work with Player. Stage is a 2.5D multiple-robot simulator[20]. Gazebo is a 3D multi-robot simulator with dynamics for outdoor environments.[21] For these tests, whenever we used Player we also used Stage since it was simpler than Gazebo and the vehicle only moves in two dimensions. USARSim is a high-fidelity simulation of robots and environments based on a commercial game engine [22]. While USARSim is itself open-source and portable, its dependence on a commercial game engine prevented its use on the system where the simulated tests were performed.

```
[PLATFORM.ATRV]
MAX_TRAN_VEL = 2.5
MAX_TRAN_ACC = 0.2
MAX_ROT_VEL = 5
MAX_ROT_ACC = 200
V_CUTOFF_ANGLE = 15
W_CUTOFF_ANGLE = 15
```

**Figure 6: Moast.ini listing with global constraints.**

The global constraints are stored in the section "[PLATFORM.ATRV]" of the file "etc/moast.ini" as shown in Figure 6. In other words, $V_{TMAX}$ = 2.5 m/s ( 8.2 ft/s) , $A_{TMAX}$ =0.2 m/s$^2$ (0.65 ft/s$^2$), $V_{RMAX}$= 0.8 rad/s (5°/s), $A_{RMAX}$= 3.49 rad/s$^2$ (200°/s$^2$), and $\theta_{VCUTOFF}$ = $\theta_{WCUTOFF}$ = 0.26 rad (15°). The parameters are left the same for all tests except for the runs for A0 where $\theta_{VCUTOFF,}$ and $\theta_{WCUTOFF}$ are set to between 10° and 25° that are indicated with the subscripts. Diagrams of the waypoints or arc-segments commands for each test are given in the results section along with the results for that set.

## 7. Results

### Single-Turn in Simulation

This test consists of only two line segments at a 90° angle both with max_speed of 0.5 m/s (1.6 ft/s) and a neighborhood of 0.1 m (0.32 ft). In the first test, the starting position lined up exactly with the first segment. All of the A0 traces overshot the corner by more than 0.3m (1ft). To bypass this issue, a small offset of 0.25 m (0.82 ft) to the starting position was added. The plots of the test with A0 running in simulation are shown in Figure 7. Plots for the other algorithms in the same test are shown in Figure 8. The metrics for the test are given in Table 1. It was surprising that A0 was faster with

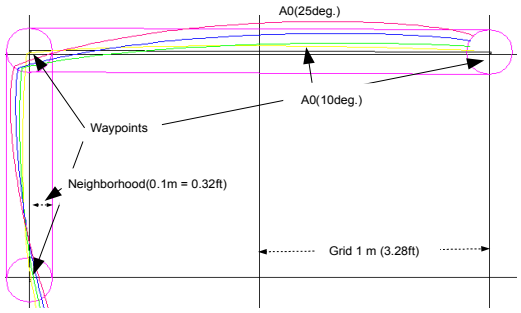


**Figure 7: Traces of the position of the Vehicle in simulation controlled by A0 with 10°(yellow),15°(green), 20°(blue), and 25°(red). Neighborhood around waypoints shown in magenta.**

smaller cutoff angles. Although smaller cutoff angles make the paths slightly shorter it also means that it will be stopped longer while doing the turn. The plot of velocities in Figure 9 provides two explanations. The shorter cutoff angles allowed the vehicle to reach higher speeds after the turn. A0 never decelerates properly at the end of the path. The shorter the cutoff angle, the more severe the sudden stop at the end.

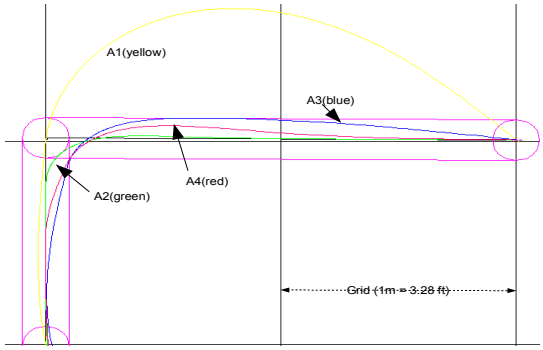


**Figure 8: Traces of the position of the Vehicle in simulation controlled by A1(yellow),A2(green),A3(blue) and A4(red)**

| Algorithm | $E_{max}$ | $T_{move}$ |
|---|---|---|
| A0(10°) | 0 | 31.14 s |
| A0(15°) | 0 | 32.55 s |
| A0(20°) | 0 | 34.79 s |
| A0(25°) | 3 cm (1.2in) | 38.29 s |
| A1 | 0.54m(1.77ft) | 33.77 s |
| A2 | 0 | 31.64 s |
| A3 | 0 | 32.74 s |
| A4 | 0 | 31.06 s |

**Table 1: Times and Error Distances for each algorithm for single turn test in simulation.**

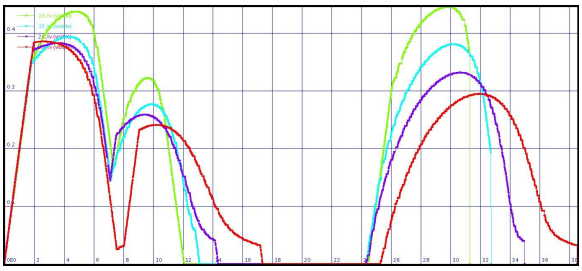

**Figure 9: Plot of the translational velocities (m/s) versus time(s) for A0 for the single turn test in simulation. A0(10°) green, A0(15°) cyan, A0(20°) blue, A0(25°) red.**

## Single-Turn on Real AGV

For experiments using a real AGV, we eliminated A1 and the A0(25°) tests that left the required neighborhood in simulation, as well as A0(10°) test that had an extremely sudden stop at the end. The position traces are shown in Figure 10. The position was produced from a commercial system using a spinning laser mounted on a post on top of the vehicle. The spinning laser detects highly reflective targets mounted on the walls and other fixed structures in the environment. The position of each of these targets was stored in the system before the tests. When the lasers detect these targets it triangulates its own position from the position of the targets. Unfortunately, the vehicle vibrates as it turns which causes the position sensor on the mast to move more than the base of the vehicle. This causes the positions recorded to be noisier than the simulation traces. The times values shown in Table 2 seem to indicate that while all of the algorithms had onger times than in simulation, the increase for A3 and A4 (which depended on the relative position of a point farther away than A2) were also slowed less than A0, which depends on angular error.

| Algorithm | $E_{max}$ | $T_{move}$ |
|---|---|---|
| A0(15°) | 3 cm (1.2 in) | 49.63 s |
| A0(20°) | 4 cm (1.6in) | 57.08 s |
| A2 | 5 cm(2.0in) | 55.06 s |
| A3 | 3 cm(1.2in) | 43.90 s |
| A4 | 1 cm(0.4in) | 41.57 s |

**Table 2: Times and Error Distances for each algorithm for single turn test or real AGV.**

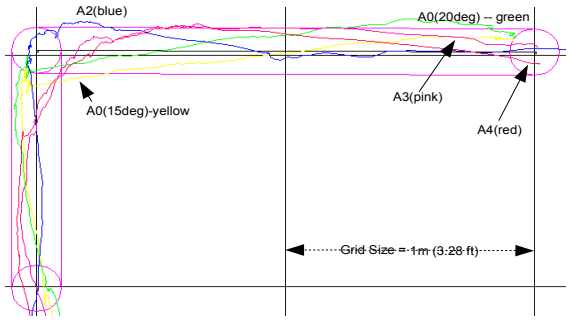


**Figure 10: Position traces recorded on the real AGV while controlled by A0(15°)-yellow, A0(20°)-green, A1-blue, A3-pink, and A4-red.**

## 8. Conclusion

MOAST provides a good framework for the evaluation of algorithms for the primitive mobility level of an Autonomous Guided Vehicle. All the algorithm developer/tester needs to do is code a function to compute the commanded velocities from the sensed/simulated positioning and commanded path with constraints and the frame work provides everything else needed to run it in simulation or on a real vehicle. The framework was recently extended with additional data logging and display capabilities for analyzing these algorithms. The simulation is useful for debugging algorithms and can isolate issues that might have been lost in the noise when running on the real system; however, testing on the real vehicle showed that an algorithm that was only slightly better in simulation was more substantially better in the real world. It would be beneficial if future algorithm developers will commit their algorithms to the MOAST repository so they can be more directly compared with those that came before.

## 9. REFERENCES

[1] Balakirsky, S. B., Proctor, F. M., Scrapper, C. J., Kramer, T., "An Integrated Control and Simulation Environment for Mobile Robot Software Development", Proceedings of IDETC/CIE 2008 ASME 2008 International Design Engineering Technical Conference & Computers and Information in Engineering Conference ,New York, NY, August 04-07, 2008, http://www.mel.nist.gov/publications/get_pdf.cgi?pub_id=824656

[2] Scrapper, C. J., Balakirsky, S. B., Messina, E., "MOAST and USARSim-A Combined Framework for the Development and Testing of Autonomous Systems", Proceedings of the SPIE Defense and Security Symposium, Orlando, FL, April 17-21, 2006, http://www.mel.nist.gov/publications/get_pdf.cgi?pub_id=822696

[3] "National Institute of Standards and Technology", http://www.nist.gov/

[4] “Real-Time Control Systems Library — Software and Documentation”, http://www.isd.mel.nist.gov/projects/rcslib/

[5] Albus, J. S., ”4D/RCS A Reference Model Architecture for Intelligent Unmanned Ground Vehicles”, Proceedings of the SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, FL, April 1-5, 2002, http://www.mel.nist.gov/publications/get_pdf.cgi?pub_id=821736

[6] “RoboCup Rescue”, http://www.robocuprescue.org/

[7] “2010 Virtual Manufacturing Automation Competition”, http://vma-competition.com/

[8] “SourceForge.net: MOAST Wiki”,http://sourceforge.net/apps/mediawiki/moast/index.php?title=Main_Page

[9] Shackleford, W. P., Bostelman, R. V., “Data Collection Test-bed for the Evaluation of Range Imaging Sensors for ANSI/ITSDF B56.5 Safety Standard for Guided Industrial Vehicles”, Proceedings of the Performance Metrics for Intelligent Systems Workshop 2009 (PerMIS), Gaithersburg, MD, September 21-23, 2009, http://www.mel.nist.gov/publications/get_pdf.cgi?pub_id=903083

[10] Mobility Open Architecture Simulation and Tools (MOAST) framework, https://sourceforge.net/projects/moast

[11] “Automated guided vehicle”, http://en.wikipedia.org/wiki/Automated_guided_vehicle

[12] Koff, G.A., Demag, R., “Automatic Guided Vehicle Systems: Basics of ATVS”, 1985 National Material Handling Forum, February 27, 1985, http://www.mhia.org/learning/resources/downloadfile.aspx?id=4731

[13] “Mobility and Manipulation Project”,http://www.nist.gov/mel/isd/ms/mmp.cfm

[14] Balakirsky, S., "MOAST Mobility Open Architecture Simulation and Tools Reference Manual”, http://sourceforge.net/projects/moast/files/reference%20manual/3.4/refManualV3.4.pdf/download

[15] Neutral Message Language (NML) , http://www.isd.mel.nist.gov/projects/rcslib/NMLcpp.html

[16] Player Project, http://playerstage.sourceforge.net/index.php?src=player

[17] NML Message Files, http://www.isd.mel.nist.gov/projects/rcslib/message_files.html

[18] Performance Simulation Project, http://www.nist.gov/mel/isd/ks/persim.cfm

[19] Real-Time Control Systems Library, http://www.isd.mel.nist.gov/projects/rcslib/

[20] Stage (part of the Player Project), http://playerstage.sourceforge.net/index.php?src=stage

[21] Gazebo (part of the Player Project), http://playerstage.sourceforge.net/index.php?src=gazebo

[22] USARSim wiki, http://usarsim.sourceforge.net/wiki/index.php/Main_Page