A Six-Wheeled Omnidirectional Autonomous Mobile Robot[1]

Kevin L. Moore, *Senior Member, IEEE*, and Nicholas S. Flann
Center for Self-Organizing and Intelligent Systems
Utah State University, Logan, UT 84322
moorek@ece.usu.edu, flann@nick.cs.usu.edu

People have long imagined vehicles and machines that direct themselves. From early science fiction literature to the 20th-century American cartoon, *The Jetsons*, to recent initiatives such as the Intelligent Vehicle Highway System, the autonomous, intelligent machine, or robotic system, has been one of the holy grails of technology. A particularly fertile field for autonomous systems research is the unmanned vehicle arena. Mobile robots have been developed that can operate autonomously in underwater environments as well as in air and outer space. More recently, there has been increasing interest in unmanned ground vehicles, or UGVs, especially for use in the military, in agriculture, and in civilian transportation. Today, advances in mechanical design capabilities, sensing technologies such as GPS (global positioning system), computing power and miniature electronics, and intelligent algorithms for planning and control have led to the possibility of realizing true autonomous mobile robot operation for UGV applications.

In this article we describe a specific autonomous mobile robot developed for UGV applications. The article focuses on a novel robotic platform [1], the Utah State University (USU) Omnidirectional Vehicle (ODV). This platform features multiple "smart wheels" in which both the speed and direction of each wheel can be independently controlled through dedicated processors. The result is a robot with the ability to completely control both the vehicle's orientation and its motion in a plane – in effect, a "hovercraft on wheels."

We begin by describing the mobility capability inherent in the smart wheel concept and discussing the distributed-processor mechatronic implementation of the complete system. Then we focus on a multiresolution behavior-generation strategy that we have developed for controlling the robot. The strategy is characterized by a hierarchical task decomposition approach. At the supervisory level, a knowledge-based planner and an A*-optimization algorithm are used to specify the vehicle's path as a sequence of

[1] This article is an expanded version of the paper "Hierarchical task decomposition approach to path planning and control for an omnidirectional autonomous mobile robot," K. Moore and N. Flann, *Proceedings of 1999 IEEE International Symposium on Intelligent Control/Intelligent Systems and Semiotics*, Cambridge, MA, September 1999.

basic maneuvers. At the vehicle level, these basic maneuvers are converted to time-domain trajectories. These trajectories are then tracked in an inertial reference frame using a model-based feedback linearization controller that computes set points for each wheel's low-level drive motor and steering angle motor controllers. The effectiveness of the strategy is demonstrated by results from actual tests with several real robots designed using the smart wheel concept.

The USU ODV Robotic Platform

Mobility Capability and Mobility Control

Much of the research described here was funded by the U.S. Army Tank-Automotive and Armament Command's Intelligent Mobility Program. The broad objective of the program has been to develop and demonstrate intelligent mobility concepts for unmanned ground vehicles. Mobility means, literally, "the ability or readiness to move or be moved; being mobile" [2]. Intelligent mobility adds the capability to determine optimal paths through terrain by using various smart or intelligent navigation and path-planning strategies (i.e., obstacle avoidance and negotiation using cost functions and tradeoff analyses). Thus, in the context of autonomous mobile robotics, we consider two components of intelligent mobility: *mobility capability,* and *mobility control.* Mobility capability refers to the physical characteristics of the vehicle that make it able to move. Mobility control means using "intelligence" in controlling the vehicle to actually achieve its full mobility capability. Mobility control also has two components:

1. First, to manage the "local" dynamic interactions between the vehicle and the forces it encounters, intelligent *vehicle-level control* algorithms must be developed and implemented to optimally maneuver the vehicle.
2. Second, *mission mobility and path planning* is concerned with "global" motion planning and navigation and is used to determine the path a vehicle should take to pass through a given region to achieve its objective.

The USU “Smart Wheel”

Our perspective on developing effective robotic systems for UGV applications is that one must have both a mobility capability to work with and the proper mobility control to effectively utilize that capability. To this end, we have developed a series of novel mobile robots based on a specific mobility capability that we call the “smart wheel.” Figure 1 shows the T1, a 95-lb ODV vehicle with six smart wheels. Other USU ODV six-wheel vehicles include the ARC III, a 45-lb small-scale robot [1] and the T2 [3], a 1480-lb robot (shown in Fig. 2). The USU smart wheel concept is shown in Fig. 3. Each smart wheel has a drive motor, power (in T1), and a microcontroller, all in the wheel hub. This is combined with a separate steering motor and with actuation in the *z*-axis (in the T3, a newly developed robot, not shown) to create a three degree of freedom mechanism. Infinite rotation in the steering degree-of-freedom is achieved through an innovative slip ring that allows data and (in the T2) power to pass from the wheel to the chassis without a wired connection.

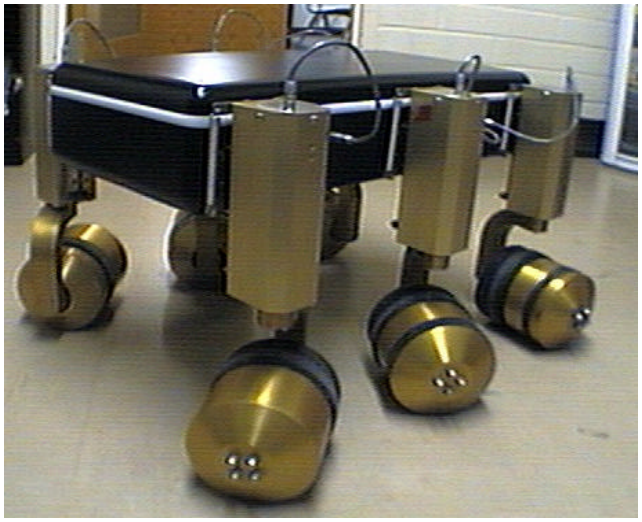

Figure 1. The T1, a USU ODV vehicle (95 lb).

Figure 2. The T2 ODV autonomous mobile robot (1480 lb).

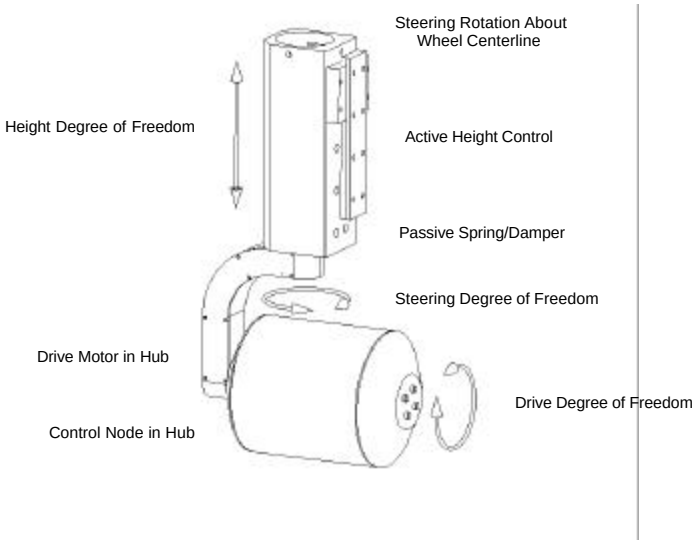

Figure 3. The USU smart wheel.

The robotic platforms resulting from attaching multiple smart wheels to a chassis are called omnidirectional because, as we show later, the resulting vehicle can drive a path with independent orientation and *X*-*Y* motion. This differs from a traditional Ackerman-steered vehicle or a tracked vehicle that must use skid-steering. In such vehicles, orientation is constrained by the direction of travel. Much of our current research focuses on exploring the mobility improvements offered by the ODV concept over more traditional vehicle steering mechanisms [4], [5]. It should be noted that the USU T-series of robotic vehicles is not truly omnidirectional. The vehicles are, in fact, nonholonomic because it takes a finite time to turn a wheel to a new steering angle. Indeed, this time introduces a coupling in the *X*-*Y* motion of the vehicle. Nevertheless, we use the term *omnidirectional* because the steering motors have a very fast turn rate relative to the dynamics of the vehicle itself, resulting in what is effectively an omnidirectional capability.

Also note that the robots shown in Figs. 1 and 2 have six wheels. This is not necessary from a mobility capability standpoint; that is, we can achieve the ODV behavior with fewer wheels. Indeed, at least three other ODV concepts have been developed by other researchers. These include three- and four-wheel ODV robots (see [6], [7], and [8]). An early smart-wheel-based USU robot also had four wheels [1]. In the case of the T1 and T2 robots, we used six wheels simply to get more power and tire surface on the ground.

Other robotic vehicles have been developed with multi-wheel steering. In some cases, such vehicles have been carefully modeled [9] and their behavior has been studied at length [10], [11]. The vehicle described here differs from other ODV robots as a result of the particular design of the smart wheel, with the slip ring that allows infinite rotation of each wheel in either direction. This design gives the robots described here distinctive mobility capability characteristics. In this article, however, we do not consider the tradeoffs between the mechanical mobility capability of the robots described here and others described in the UGV literature. Instead the contribution is the integrated planning and control strategy we have developed for controlling the tasking and execution of the USU T-series of robots. This strategy combines a task-based trajectory planning strategy with a first-principles-derived, model-based controller developed specifically to exploit the mobility capabilities of our specific robot.

Vehicle Electronics

Overall mobility capability is provided through a complete mechatronic system that must include a mechanical concept (the independent steering and drive mechanism) and suitable vehicle electronics and control systems to properly actuate the mechanical subsystem. The vehicle electronics system (called

"vetronics" in the UGV community) used on the T-series ODV robots is what enables the algorithms for controlling the robot to actuate the mechanical capability provided by the smart wheel concept. Fig. 4 shows the vetronics architecture on the T2 vehicle. The system is built around three single-board Pentium computers running Linux and communicating via TCP/IP using a local area network (LAN) on the vehicle. User interfaces to the vehicle are through a wireless modem (for the joystick) and a wireless TCP/IP link for talking to a graphical user interface (GUI). The processors communicate with various sensors (such as GPS and a fiber-optic gyro) and with other parts of the system in several ways (including CAN bus, A/D and D/A conversions, and RS-232, on both PCI and PC-104 buses). In particular, the master node PC talks to six different wheel node electronic units, one for each wheel. Each wheel node includes a 32-bit microcontroller with interfaces to a variety of sensors, feedback from both absolute and incremental encoders, and a PWM-based driver for switching up to 60 amps per wheel on a 48V bus (see Fig. 5). Add to this the actual motors for drive and steering, the power distribution system, and the actual software required to implement control algorithms and planning and decision-making logic, and you have a very complex system that must be designed in a highly-integrated way to work properly.

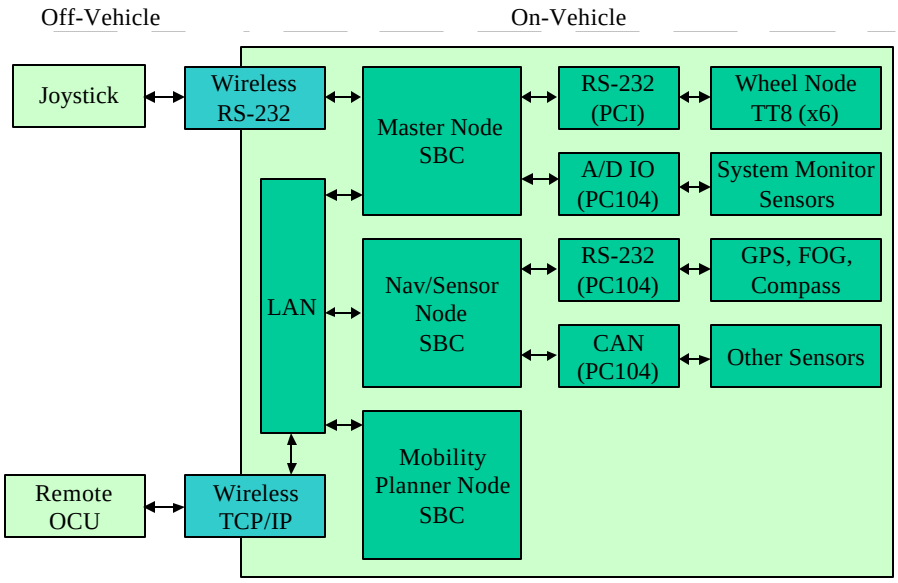

Figure 4. T2 vehicle electronics architecture.

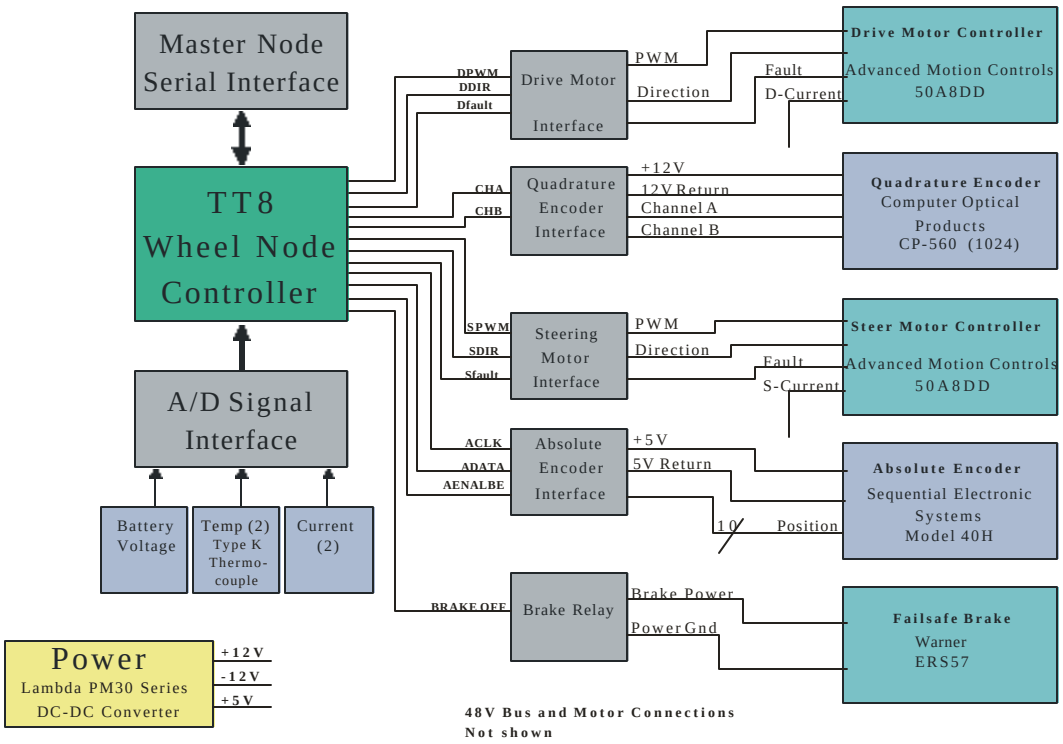

Figure 5. T2 wheel node vetronics architecture.

Integrated System Engineering and Functional System Architecture

The smart wheel is used as the central building block for putting together a complete autonomous mobile robot. The other components in the system are shown in Fig. 6. Starting with the smart wheel and the smart running gear (another term for the z-axis actuation that will appear in the T3, a new version of our robot currently under development), we add appropriate vehicle electronics, as described in the previous section. The result is a physical system capable of implementing algorithms. The final task is to add these algorithms, first at the lower level, for vehicle-level control, and then at the higher-level, for vehicle tasking control. All of these pieces are then tied together, resulting in the complete system.

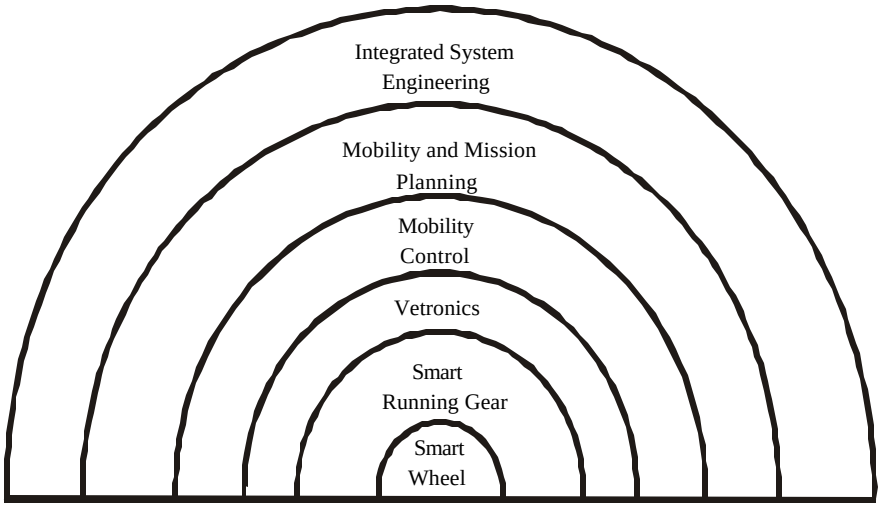

Figure 6. System engineering (figure thanks to George Powell of VPI, Inc.).

The overall functional system architecture used to command and control the vehicle is shown in Fig. 7. The system has three distinct modes of operation: manual control, closed-loop control, and exception control. In the manual control mode, an operator uses a joystick to maneuver the vehicle. A radio modem is used for communication with the joystick. In manual control, commands from the user (movements of the joystick encoded as voltages) are translated into body-referenced motion commands ($\dot{x}, \dot{y}, \dot{q}$) by the joystick interpreter. In the closed-loop control mode, the mission and mobility planner, under high-level direction from a user, controls the vehicle. In the exception control mode, the vehicle-level controller may take actions as required to deal with unexpected situations. Such actions might be a vehicle shutdown, a reactive behavior to avoid an unplanned obstacle in its path, or a decision to request that the planner execute a new plan. This module is the subject of ongoing research and our current implementation of the exception control module simply shuts down the vehicle if it gets too far off its path or if various communications systems exceptions occur. However, although not the focus of this article, in the long run, the exception control block will be the true heart of the intelligent mobile vehicle.

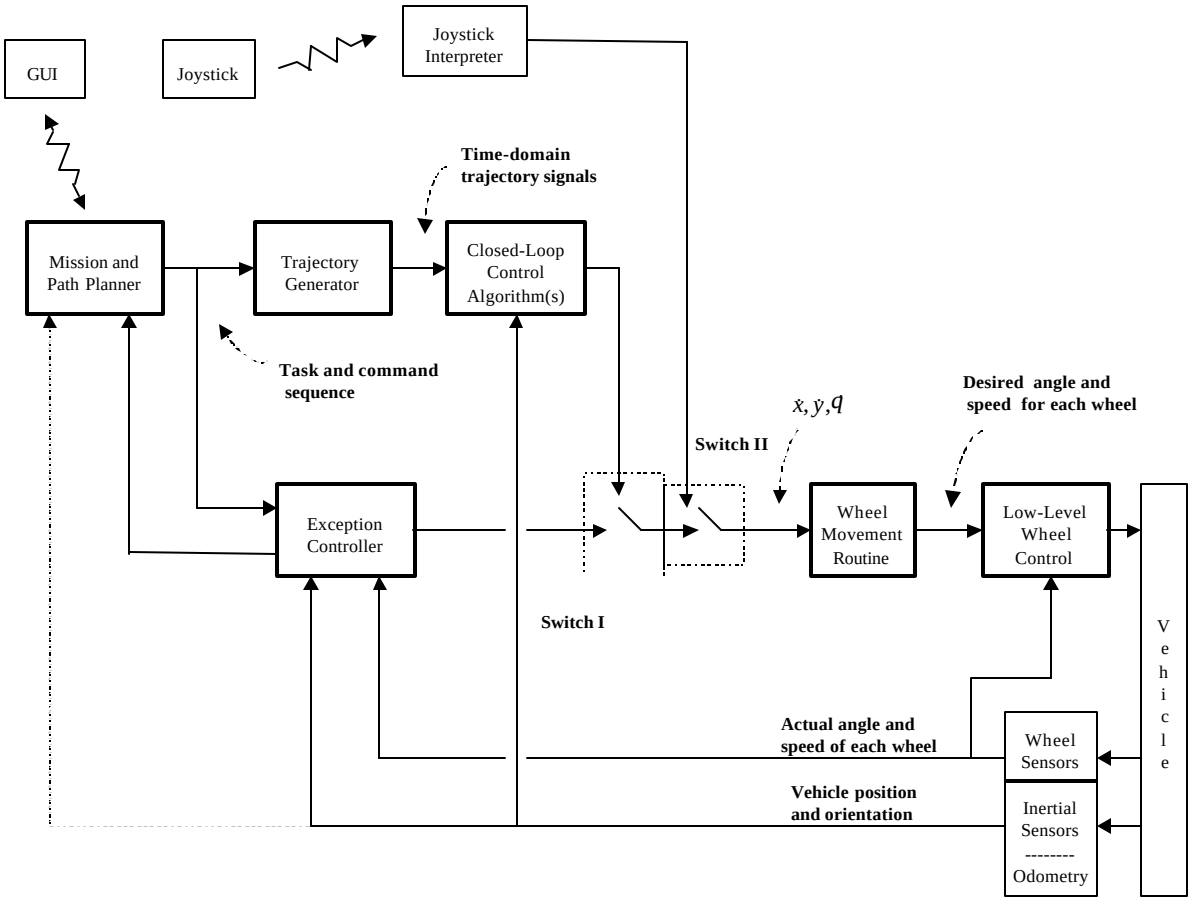

Figure 7. Control system architecture.

Task Decomposition Approach to Controlling the ODV

Given the mobility capability described in the previous section, we can now describe the mobility planning and control strategies used on the T-series of ODV vehicles. The normal operating mode for the vehicle is the closed-loop control mode. This mode is used to follow a prescribed inertial path using feedback of the vehicle's actual position and orientation, obtained via GPS or odometry. A three-level, hierarchical control scheme is used that results in a multiresolution behavior-generation strategy. The hierarchical task decomposition approach described here is common in motion-generation problems for robotic manipulators in manufacturing settings [12], [13], and in applied artificial intelligence approaches to planning for mobile robotics [14], [15], [16].

Multiresolution Architecture

Fig. 8 shows the multiresolution signal flow used by our system in closed-loop mode. At the coarsest resolution, the strategy is characterized by a task decomposition approach in which an object-oriented, knowledge-based planner and an A*-optimization algorithm [17] are used to specify the vehicle's path as a sequence of basic maneuvers. At medium resolution, or the vehicle command level, the basic maneuvers are converted to time-domain trajectories and a nonlinear controller is used to generate low-level set points to force the vehicle to track the required path in an inertial reference frame. At the finest resolution, classical control is used to drive the vehicle's wheels to the required speed and angle set points.

Note that each loop has a different bandwidth. At the lowest level, or finest resolution, the controllers for the individual wheel drive and angle motors operate at the highest bandwidth (approximately 10 Hz). At medium resolution, the path-tracking control algorithms operate at a middle-range bandwidth (approximately 5 Hz). Finally, the path planner operates at the coarsest resolution and, correspondingly, the lowest bandwidth (on the order of 0.1 Hz).

In terms of information flow in the system we have increased abstraction with decreasing bandwidth. In the low-level controller we generate voltage and current commands to send through D/A converters to the vehicle actuators (motors). Information sources at this level are encoder outputs that describe the motor positions and speeds. At the next level up in the hierarchy, the path-tracking controller communicates motor set points to the low-level controller over an RS-232 link, using an appropriate protocol for the digital communications. Information feedback at this level includes GPS signals and the output of odometry calculations, which are more abstract signals than the encoder information used by the low-level controllers. Finally, the information passed from the planner to the path-tracking controller takes the form of a script that has encoded desired vehicle motions into a sequence of parameterized commands communicated in ASCII using the TCP/IP protocol. Information passed back to the planner includes vehicle status information (position, orientation), and unexpected obstacle polygons.

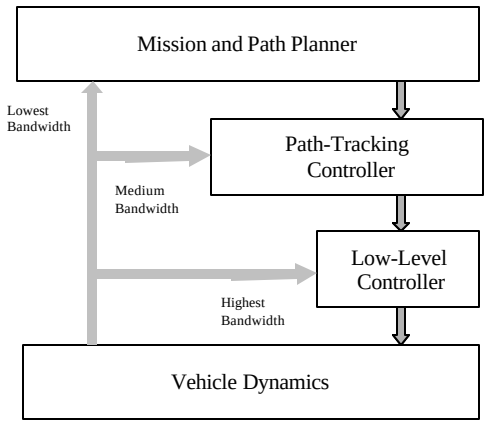

Figure 8. Multiresolution interpretation of control system architecture.

Task Decomposition Approach to Behavior Generation

To describe how the multiresolution architecture shown in Fig. 8 is used to generate vehicle behavior by decomposing these behaviors into basic tasks, we consider the basic steps involved in a typical session with the vehicle.

Step 1: Task Selection. First, a user interfaces with the mission and path planner via a graphical user interface (GUI) to select desired goal tasks by indicating location points or regions on a map. The session begins with the user choosing a map. The map may be obtained from aerial photographs or satellite images and must have been calibrated with GPS before it can be used. Then the user defines the mission goals. Two types of goal tasks can be chosen in our current system: visit goal tasks or sweep goal tasks. Visit goals are simply a set of locations to which the vehicle must travel. Sweep goals require the vehicle to follow a path that will take it through a given region, assuming the vehicle "sweeps out" a given area as it travels. Fig. 9 shows a GUI in which an operator has used a mouse to define a sweep region.

Figure 9. Sweep goal task.

Step 2: Path Planning. After the user has defined the goal tasks, the mission and path planner then determines a path for the vehicle that optimizes the achievement of the goal tasks. This path is defined by a sequence of commands selected from a maneuver grammar, or set of basic motion primitives for the vehicle. Some of these maneuver primitives are shown in Fig. 10. Fig. 11 shows the path produced by the planner for the sweep region indicated in Fig. 9. To produce this path the planner assumed that the vehicle was pulling an implement, such as a harrow or a sprayer that would sweep out a given area, and then, based on this assumption, divided the region with a path that is a series of translates separated by curves. In the next section, we describe the process of generating a sequence of commands in more detail.

Step 3: Start Mission. After the mission and path planner has generated a script of basic maneuvers and has displayed the path on the GUI, the user may start the mission from a pull-down menu on the GUI. The vehicle then attempts to follow the path defined by the mission and path planner. Fig. 11 shows how the user would see the progress of the vehicle execution indicated on the GUI. Functionally, vehicle execution involves the next three steps.

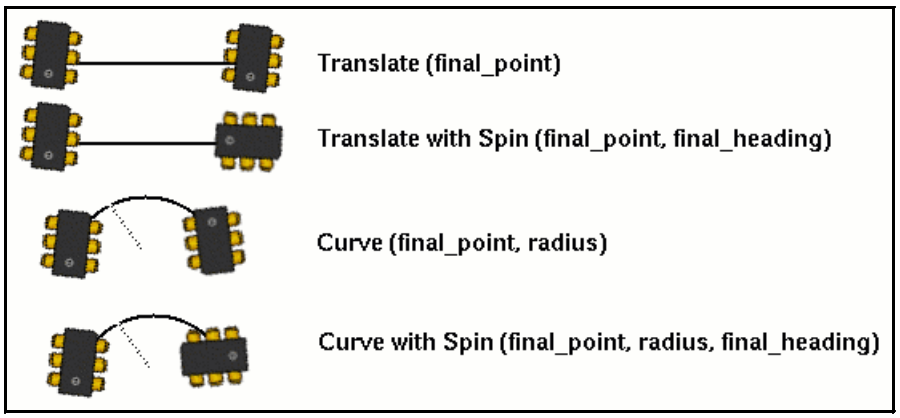

Figure 10. Maneuver grammar.

Figure 11. Path generated for the sweep goal task shown in Fig. 9.

Step 4: Trajectory Generation. After a path has been determined and vehicle execution is initiated, the planner sends the script of basic maneuvers (called "z-commands" in our software protocol) to the vehicle-level controller on the master node (see Fig. 4). The script is then processed by the trajectory generator (see Fig. 7), which converts the z-commands into time-domain signals, $x(t), y(t), \boldsymbol{q}(t)$. These signals define x-axis, y-axis, and yaw rotation reference trajectories, respectively, for the desired vehicle motion. For example, Fig. 12 shows a sample path in inertial coordinates. This path could be described by a script, or sequence of commands, such as:

1. Translate at 45 deg with a speed of $\sqrt{2}$ ft/sec for 20 sec.
2. Translate at 90 deg with a speed of 1 ft/sec for 20 sec.
3. Translate at 0 deg with a speed of 2 ft/sec for 20 sec.
4. Translate at –90 deg with a speed of 1 ft/sec for 20 sec.
5. Translate at –180 deg with a speed of 0.5 ft/sec for 20 sec.

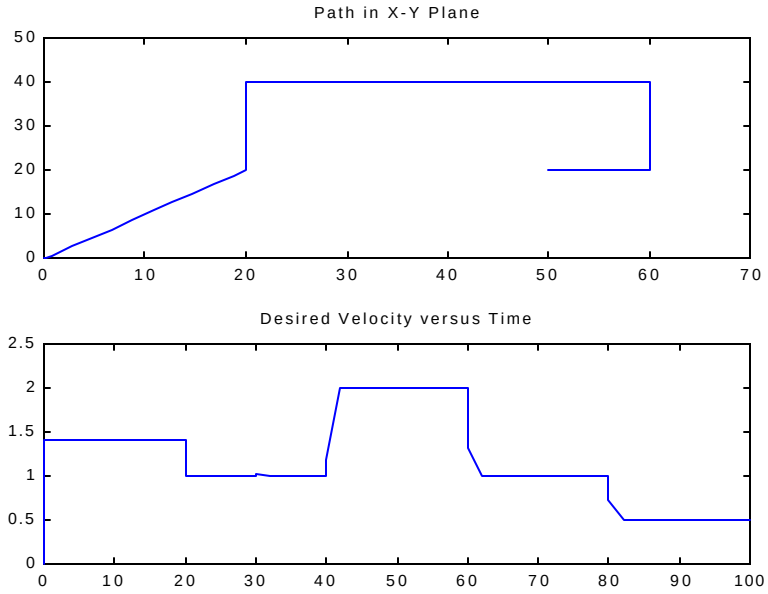

Figure 12. Sample trajectory in inertial space.

The trajectory generator parses and filters this script to produce time-domain signals, taking care to preserve the continuity of position and velocity at the endpoints between maneuvers. Fig. 13 shows time domain signals for position produced by the trajectory generator for the script corresponding to Fig. 12. Note that in our implementation, the trajectory generator decomposes the sequences of basic maneuvers into time functions that are linear combinations of steps, ramps, decaying exponentials, and sinusoidal functions. This simplifies controller design for tracking at the next level in the hierarchy.

Step 5: Path-Tracking Control. Next, the time-domain signals computed by the trajectory generator are used as reference signals by the closed-loop controller, which computes appropriate corrections to the vehicle heading (in body coordinates). These vehicle heading corrections are passed to the wheel movement routine (see Fig. 7), which uses kinematic relationships to generate set points for the low-level controller.

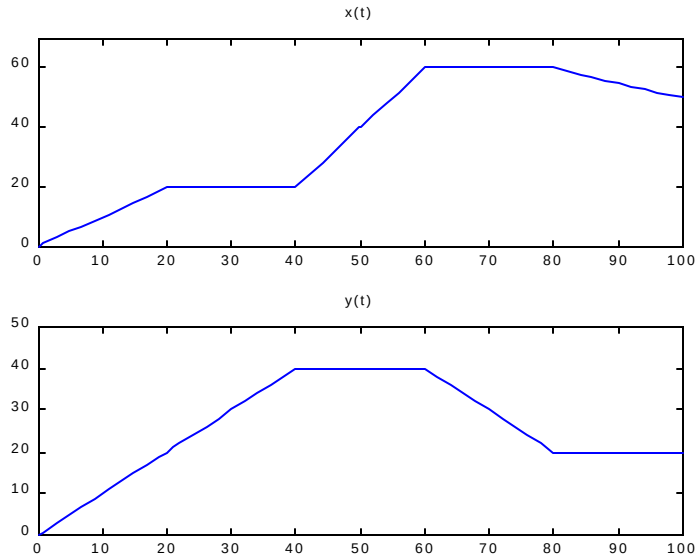

Figure 13. Corresponding time-domain trajectories.

Step 6: Mission Execution. Finally, the vehicle execution is monitored and controlled by the exception control block (again, see Fig. 7), which uses broad constraints to decide if it is necessary to intervene in the overall control system execution. This block monitors vehicle performance, provides low-bandwidth feedback to the mission and path planner, and executes emergency or "exception" maneuvers (e.g., invoking a replan event in the mission and path planned, carrying out a rocking procedure to get out of a rut, or executing an emergency shutdown). This block contains considerable logic that is formulated in truth tables and executed as a finite state machine. As noted above, the current state of this block is to simply detect various forms of vehicle malfunction, such as a tracking error that is too large or a communication failure. However, it is interesting to note that the net effect of this logic is to make the vehicle a hybrid system, supervised by a logic controller that initiates discrete-event state transitions, which result in the execution of continuous dynamic system behavior. Ongoing efforts are aimed at exploring this hybrid system interpretation of our vehicle.

Mission and Path-Planning Algorithms

The mission and path planner component of the overall system solves the following problem:

Given:

An approximate terrain map of the environment.

An omnidirectional vehicle.

A stream of goal locations to visit and/or polygon regions to sweep.

A stream of map updates due to vehicle sensors.

Find:

An assignment of a mobility path to the vehicle.

Such That:

The goals are satisfied in the shortest time.

The mission planner is run once when the map and initial goals are entered, then again whenever new goals are entered during mission execution or when sensors provide significant changes to the initial map. The system differs from traditional path planning systems [18] in that the omnidirectional vehicle controlled require not only paths to follow, but also maneuvers to execute while following the paths. The approach here is to identify possible maneuvers as a set of parameterized commands and then specify paths as a sequence of instantiated commands. For example, since the omnidirectional vehicle is capable of controlling its body orientation independent of its direction of travel, one command (called traverse) specifies travel along a straight path while maintaining a fixed vehicle body orientation. Another command (called traverse-with-spin) specifies travel along a straight path while rotating the body orientation through a given angle. Other commands include curve-with-spin, spin-in-place, and curve-with-fixed-orientation. The task of the planner is to determine the optimal command sequence for the vehicle such that the goals are achieved in the shortest time.

The planner uses an object-oriented knowledge-based approach to determine suitable command sequences that best utilize the mobility capabilities of the omnidirectional vehicle. The 3-D terrain map is first parsed into a covering of abstract functional terrain objects, which can be linear (such as ditch, curb, ridge-line, crest, hill-toe, road, or track, etc.) or polygon (such as hill-face, field, room, flat, or obstacle, etc.). This parsing process is illustrated in Fig. 14 for a simplified example. The maps that describe the terrain are currently build manually through integration of ARC-INFO G.I.S files and elevation data, however, fully automated procedures that integrate methods from elevation map processing [19] with existing G.I.S representations are under development.

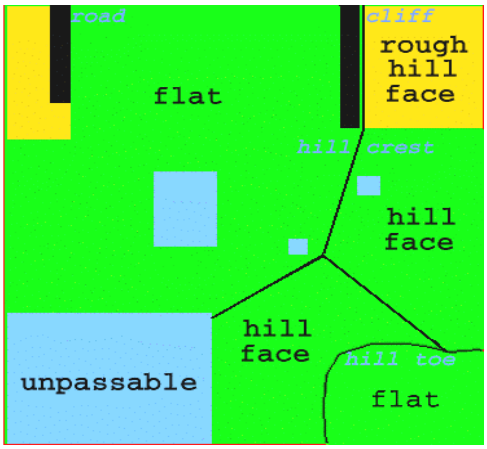

Figure 14. Functional terrain objects.

The terrain objects contain knowledge about how to traverse through the object, reach a location within the object, cross over the object, follow along the object, avoid obstacles within the object, and how to best enter and exit the object. This knowledge consists of methods that take functional terrain objects and vehicle parameters as inputs and compute alternative mobility paths (sequences of maneuver commands). Fig. 15 shows the class hierarchy used in our planning system.

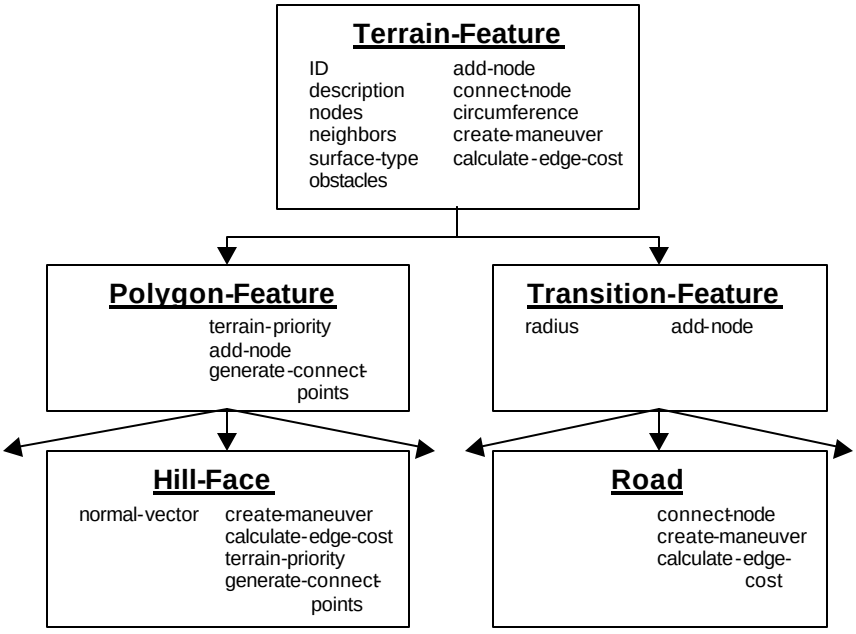

Figure 15. Class hierarchy for terrain objects.

Next, each object identified in the map uses its methods to compute a set of “reasonable” mobility paths that connect a network or graph linking the location of the vehicle with the locations of the goals. An

example of such a graph is illustrated in Fig. 16. The edges of the graph link the goal locations (denoted by boxes) with each other and also use the existing infrastructure of the roads (refer again to Fig. 14) while avoiding the known obstacles. The edges that traverse the hill-face object form "zig zag" paths – sequences of traverse maneuvers that enable steep slopes to be climbed by switching back and forth up the hill while maintaining the long axis of the vehicle in an uphill orientation. Each edge in the graph is assigned a cost based on the estimated time required to traverse the edge by the omnidirectional vehicle. These costs are now propagated through the graph using an incremental variant of the shortest-path algorithm [20] to determine estimated costs between the vehicle location and the goals and between the goals themselves.

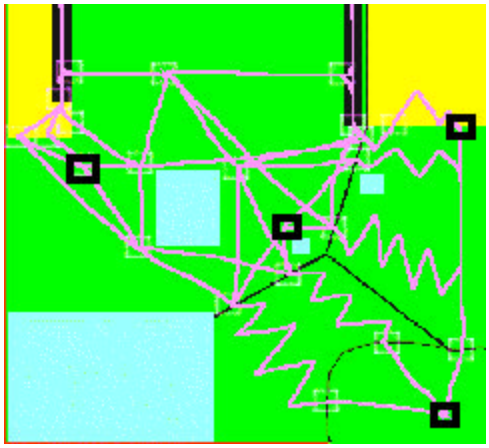

Figure 16. "Reasonable" paths through terrain objects.

A traveling salesman problem (TSP) problem is then solved using a heuristic method to determine the order to visit the goals, such that the total time is minimized. The heuristic search engine quickly identifies a near-optimal ordering of the goals for the vehicle by a best-first branch-and-bound process. A global bound is maintained that keeps track of the best solution cost so far, where the solution cost is defined as the duration of the vehicle tour. If any partial solution exceeds that bound, it is terminated. The quality vs. time tradeoff is controlled by limiting the total number of partial solutions expanded. Experimental studies have found that limiting this number to 10,000 leads to high-quality solutions within 1-2 sec of CPU time (depending on the number of goals). The search algorithm takes a partial solution and a list of remaining goals, initialized as empty and a list of goals respectively. A list of new partial solutions is formed by appending each of the remaining goals to the current partial solution. This list is then sorted

lowest first, then each new solution is then recursively expanded. Because the individual solutions are expanded locally best first, the first solution identified will be locally optimal and provide a tight bound for the branch-and-bound algorithm.

The final stage is to use an A*-algorithm to identify the best sequence of graph edges and corresponding mobility commands for the vehicle that will take it through the assigned goals. A local graph search is performed that starts at the vehicle's location, passes through each goal in the tour, and terminates at the assigned node. The **g** function used during the A* search is the sum of the individual edge costs in the path so far, plus any cost incurred during edge transitions. Edge transition costs can be used to take into account delays incurred by the need to change the vehicle configuration. The **h** function is the estimated cost of the remaining tour, without transition costs. The result of this A* search is that the vehicle is assigned a sequence of graph edges that take the vehicle from its current location, through its assigned goals, to its terminal node.

The final sequence of mobility commands is formed by looping through the assigned edges and appending the command sequence from each edge. It is at this final stage that "peephole" optimization is performed to identify smooth transitions between the command sequences on each edge traversed. Such optimization is needed because the mobility commands for the edges have been formulated independently within each terrain object. For example, consider the case when we have two adjoining objects, a Hill-Toe and a Flat, connecting the graph through a node N. The Hill-Toe object will start the command sequence for the edge-leaving node N and entering the hill with a spin-in-place command to correctly orient the vehicle. The adjoining Flat object that adds the edge connecting to N will add a simple Translate command to the edge. The peephole optimizer will see the Translate and Spin-in-Place adjoined and replace them with a shorter Translate followed by a Translate-with-Spin that will enable the vehicle to flow through the maneuvers without having to stop and spin. The final optimized command sequence is then transmitted to the path-tracking controller via a TCP/IP connection. Fig. 17 shows the outcome of the process for the mobility graph shown in Fig. 16.

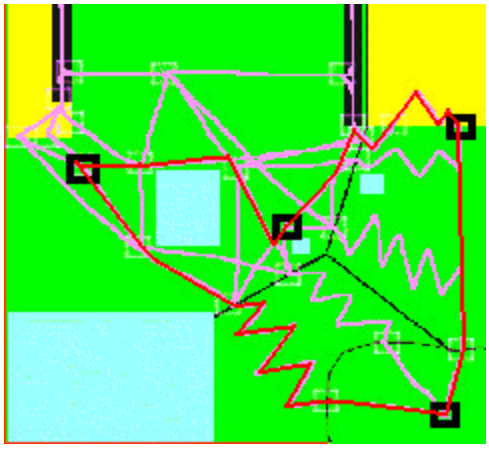

Figure 17. Final path (shown in red) chosen by the planner for the mobility graph in Fig. 16.

The mission planner continuously monitors the progress of the vehicle as the mission is executed and replans portions of the mission if needed. Events that trigger a replan include:

1. A new goal entered by the user during mission execution.
2. A map update, such as an unexpected obstacle, that prevents the current mission from being implemented.
3. The vehicle getting lost or being unable to complete its assigned goals.

Whenever a replan event is triggered, the graph is first dynamically extended to provide additional alternative mobility paths to the vehicles that overcome the detected problem, such as linking in the new goal, avoiding the obstacle, or providing better connectivity. Then the path costs among the goals and the vehicle are incrementally recomputed using an algorithm similar to D* [21], adapted for sparse graphs. Next the tour is recomputed using the updated costs and the new location of the vehicle. Finally, mobility path for the vehicle is determined and reassigned.

Path-Tracking Control

The vehicle-level control algorithms are concerned with the problem of tracking a desired vehicle motion (*x, y,* and yaw, denoted as θ, shown in Fig. 18). The path-tracking and low-level control algorithms have four key parts: the trajectory generator, the closed-loop control algorithms, the wheel movement routine, and the low-level controllers themselves. These were shown separately in Fig. 7. We have

previously described the trajectory generator, which is used to formulate the proper reference signals that the vehicle must track to achieve the desired path. These reference signals are used by the *closed-loop control algorithm* block, which compares the desired trajectory to the actual trajectory (in inertial space) and computes appropriate signals $\dot{x}, \dot{y}, \dot{q}$ in inertial coordinates to be used by the wheel movement routine. This computation is done on the basis of errors in the vehicle's inertial position. Vehicle motion is determined through either odometry or GPS measurements. Finally, the *wheel movement routine* block uses the kinematic relationships of the vehicle to translate $\dot{x}, \dot{y}, \dot{q}$ into separate drive (speed) and steering angle (direction) commands, v_i and q_i, respectively, for use by the low-level controllers. After the path-tracking controller generates appropriate speed and steering commands, proportional-integral-derivative (PID) *wheel-level controllers* are used to actually force the wheels to the correct speed and steering angle. Encoder feedback is used to compute error signals for the PID controllers. Control at this level for the ODV vehicle, including adaptive and nonlinear control algorithms, is discussed in [22]. In the remainder of this section, we focus on the closed-loop control algorithm and the wheel movement generation routine, which are also described in more detail in [23].

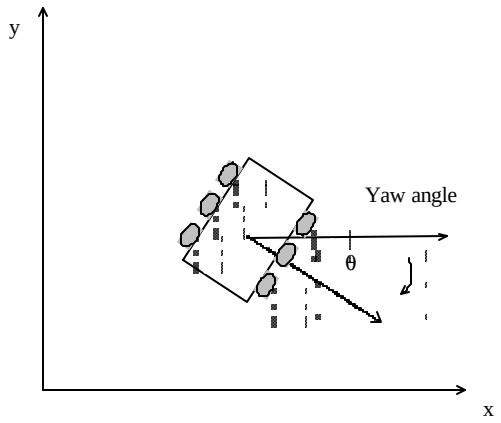

Figure 18. x, y, and yaw definitions.

Vehicle Model

To develop a control scheme, we need a reasonable model. For the ODV robot, we obtain a model by applying Newton's laws to the vehicle. This requires proper enumeration of the forces acting on the vehicle in a consistent coordinate frame. The coordinate system definition is shown in Fig. 19; the forces used in our model are shown in Fig. 20. In these figures, note especially the angle a_i, which represents the

difference between the direction the wheel is pointing and the direction it is traveling. When this angle is zero, we say that the vehicle is in kinematic equilibrium. When it is non-zero, the wheel will experience a side-slip force.

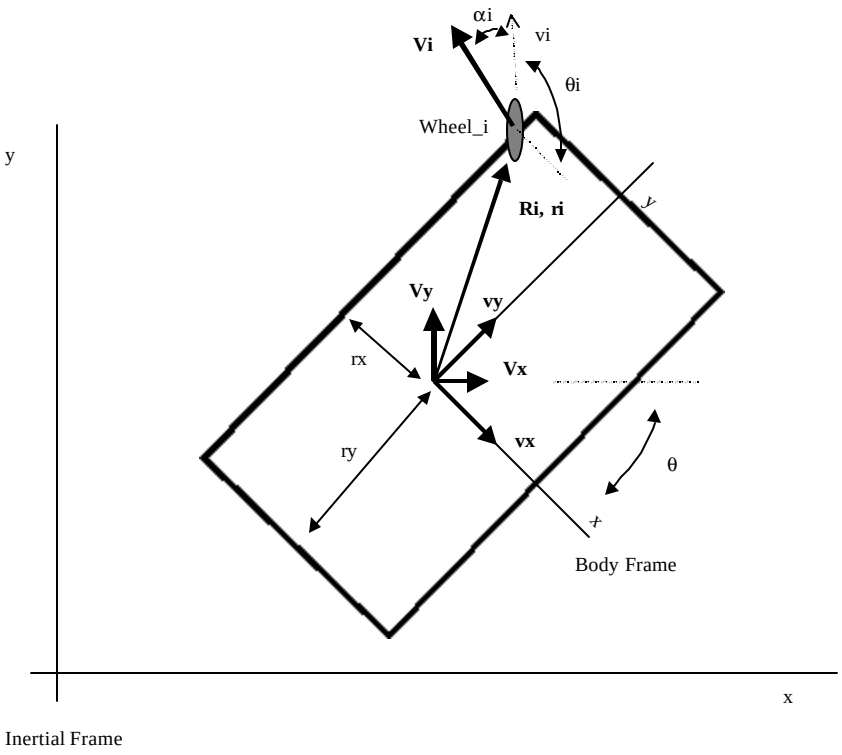

Figure 19. Coordinate systems for controller design.

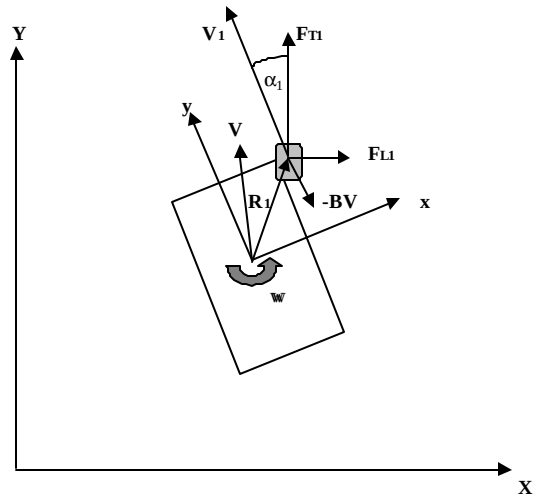

Figure 20. Forces acting on the vehicle.

In our model, we consider three key forces acting on the vehicle:

1. The force F_{Ti} is the tangential force due to the motor's torque acting to turn the wheel. This force acts in the direction the wheel is pointing.

2. Acting in a direction to oppose motion is a standard global energy dissipation force proportional to the vehicle velocity and acting in the opposite direction:

$$F_{di} = -BV_i$$

where B is the global energy dissipation coefficient, assumed to be the same at each wheel. This is basically a viscous friction force that changes from one surface to the other, but it also includes the motor losses.

3. We also include a lateral slip force F_{Li} acting to restore the vehicle's inertial velocity to match that of the wheel. This force acts 90 deg counterclockwise from the direction the wheel is pointing and has a magnitude given by

$$\left|F_{Li}\right| = \frac{mgm_s}{6}(1 - e^{-t\alpha_i})$$

where m is the vehicle mass, g is the gravitational constant, α_i is the angle mismatch between the kinematic velocity and the wheel velocity, m_s is the static coefficient of friction for the surface, and t is a parameter that depends on the surface [24]. Notice that if $\alpha_i = 0$, then $F_{Li} = 0$, so that when the vehicle is in what we call "kinematic motion," meaning the actual inertial velocity of the vehicle body at each wheel location matches the wheel's velocity, there is no side-slip force.

Using these three forces, we may write the vehicle dynamics as

$$m\begin{pmatrix}\ddot{x}\\ \ddot{y}\end{pmatrix} = \sum_{i=1}^{6}(F_{Ti} + F_{Li} - BV_i)_{inertial}$$

$$I_z\ddot{q} = \sum_{i=1}^{6} R_i \times (F_{Ti} + F_{Li} - BV_i)_{inertial}$$

where I_z is the vehicle's moment of inertial about the z-axis and R_i is the distance from the vehicle's geometric center to each wheel (see Figs. 19 and 20). Next, defining the vehicle's center of mass translational motion as $V = \begin{pmatrix}\dot{x} & \dot{y}\end{pmatrix}^T$ and using the basic kinematic relationship $V_i = V + \omega \times R_i$, we can write the equations of motion as (introducing additional notation to designate the x and y components of the forces acting at each wheel)

$$\begin{bmatrix} m & 0 & 0\\ 0 & m & 0\\ 0 & 0 & I_Z\end{bmatrix}\begin{pmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{q}\end{pmatrix} = \begin{bmatrix} -6B & 0 & 0\\ 0 & -6B & 0\\ 0 & 0 & -Br\end{bmatrix}\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{q}\end{pmatrix} + R(q)G\begin{pmatrix} F_{T1\_x} + F_{L1\_x}\\ F_{T1\_y} + F_{L1\_y}\\ \vdots\\ F_{T6\_x} + F_{L6\_x}\\ F_{T6\_y} + F_{L6\_y}\end{pmatrix}_{body}$$

where $r = 6r_x^2 + 4r_y^2$ (see Fig. 19) and $R(\theta)$ is the pseudo-Euler transformation from body to inertial coordinates, given by

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}_{inertial} = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}_{body} = R(\theta) \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}_{body}$$

Further, if we define

$$q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \quad u = \begin{pmatrix} F_{T1\_x} \\ F_{T1\_y} \\ \vdots \\ F_{T6\_x} \\ F_{T6\_y} \end{pmatrix} \quad d(\alpha, q) = \begin{pmatrix} F_{L1\_x} \\ F_{L1\_y} \\ \vdots \\ F_{L6\_x} \\ F_{L6\_y} \end{pmatrix}$$

we can see that the system has the form of a typical robotic system:

$$M\ddot{q} + B\dot{q} = R(q)G(u + d(\alpha, q))$$

where M is the mass/inertia matrix, B represents viscous damping, we have defined $R(q)=R(\theta)$, and G is a constant matrix that depends on the vehicle geometry, given by

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ -r_y & -r_x & -r_y & r_x & 0 & -r_x & 0 & r_x & r_y & -r_x & r_y & r_x \end{bmatrix}.$$

Path-Tracking Control Strategy

Given our model of the vehicle, it is easy to devise a path-tracking control strategy. First, note that $R(q)$ is invertible. Also note that G can be "inverted" using its pseudo-inverse and that the resulting product of G times its pseudo-inverse is diagonal. Thus, if we define our control signal as

$$u = G^T (GG^T)^{-1} R^{-1}(q)\bar{u}$$

the resulting system now looks like

$$M\ddot{q} + B\dot{q} = \bar{u} + R(q)Gd(\alpha, q)$$

where both *M* and *B* are diagonal. This control law is effectively an exact feedback linearization strategy [25]. However, unlike most feedback linearization schemes, in this case we do not experience a robustness problem, as there are no physical parameters in *R(q)* and there is no uncertainty in the matrix *G*, which depends only on vehicle geometry.

Next, observe that *d(0,q)=0.* Thus, if our wheels are lined up with the inertial velocity, there is no slip force. In [23] we discuss the fact that if the system is stabilized when *d(0,q)=0,* the system will be stable when the side-slip force is not zero, and the controller will in fact drive the "disturbance" *d* to zero. Thus, the final part of our controller is to simply let

$$\bar{u} = C_o(s)(q_{sp} - q) + \dot{q}_{sp}.$$

Fig. 21 shows the overall control architecture for path tracking. In this figure, *P(s)* represents the plant and the low-level PID controllers are represented by *C(s)*. Notice that we have added a feedforward term, $\dot{q}_{sp}$, which is simply the final expected velocity for the desired trajectory. This can be shown to be a computed-torque term. A final point about the architecture in Fig. 21 concerns the block G^T. This block corresponds to the block labeled "Wheel Movement Routine" in Fig. 7. This function is what translates a desired body-centered velocity vector into wheel motions (angle and speed) and is described in more detail in [23].

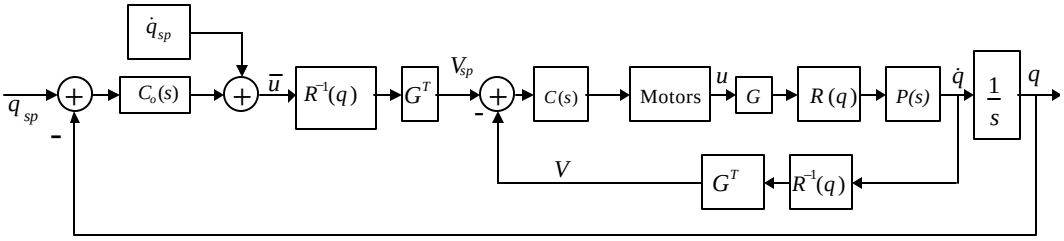

Figure 21. Vehicle-level path-tracking control architecture.

Simulation and Experimental Results

A MALTLAB/Simulink™ simulation was built to facilitate controller development. Fig. 22 shows a representative simulation of a hard closed-loop path-tracking task. Notice that in addition to a prescribed path, there are variable velocity conditions along the path, as shown in the figure, as well as a prescribed ramp up and down in vehicle orientation. Because the open-loop system seen by the outer-loop controller system looks like a single integrator with a unity-gain low-pass filter, the controllers were chosen (based on the model parameters for the corresponding surface) to be proportional-integral (PI) algorithms. This allowed us to achieve zero steady-state error to ramp inputs. It can be seen from the figure that, in simulation, the controller was effective in forcing the dynamic vehicle to follow the desired path. Though not shown in the figure, we note that the simulation is also useful in helping predict the required magnitude and rate of excursions of the wheel drive and steering motors.

To validate our controller design and simulation results, Fig. 23 shows an actual vehicle motion resulting from executing a similar trajectory to that simulated in Fig. 22 (the trajectories in the two figures have the same form, but differ in the time of the commanded task). These graphs are plots of measurements taken from a fiber-optic gyro (FOG) and from odometry calculations. The three horizontal

lines on the plot *x* versus *y* are due to lost packets in our data collection process, which was carried out over a wireless LA N. The vehicle did not make the excursions shown. Comparing Figs. 22 and 23, we see that both the simulation and the actual experiment exhibit zero steady-state error in their spatial coordinates and have essentially the same overdamped responses. With respect to the vehicle orientation, we see that the actual vehicle had some "wobble" in its motion, which is due to the precision of the FOG. This is an effect that is not modeled in our simulation. However, we see that the actual yaw trajectory achieved by the vehicle closely follows the desired ramping actions. These particular experimental results reinforce the point that our ODV robot can achieve independent control of vehicle orientation and position in the *x*-*y* plane. Other results showing the overall effectiveness of our planning and control system strategy have been demonstrated in numerous experimental tests with both the T1 and the larger T2 vehicle.

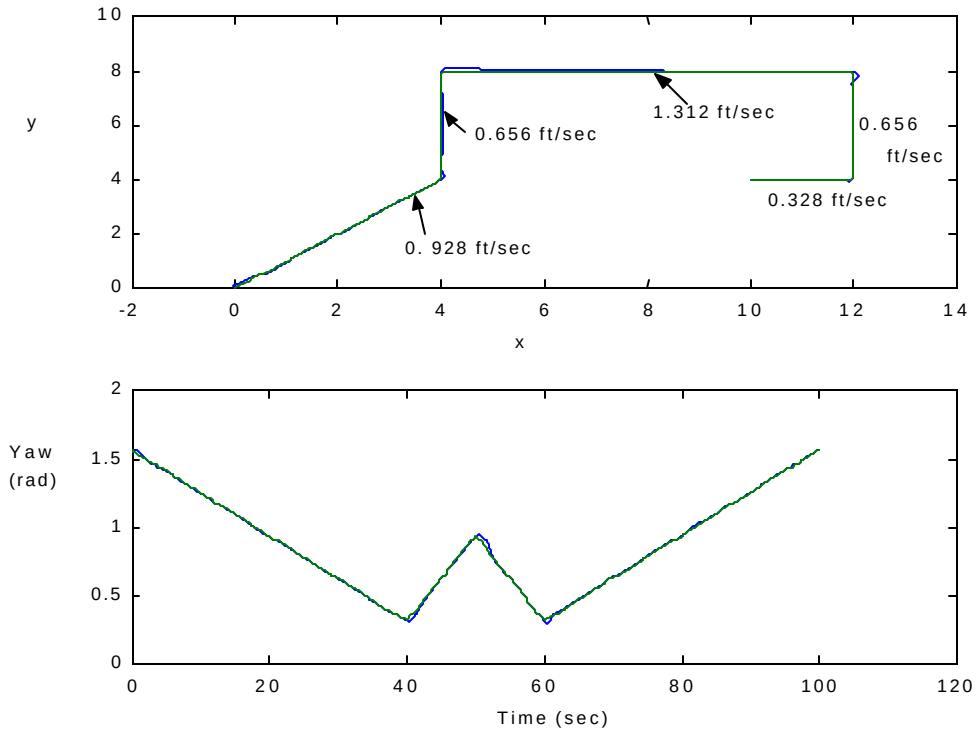

Figure 22. Simulation results.

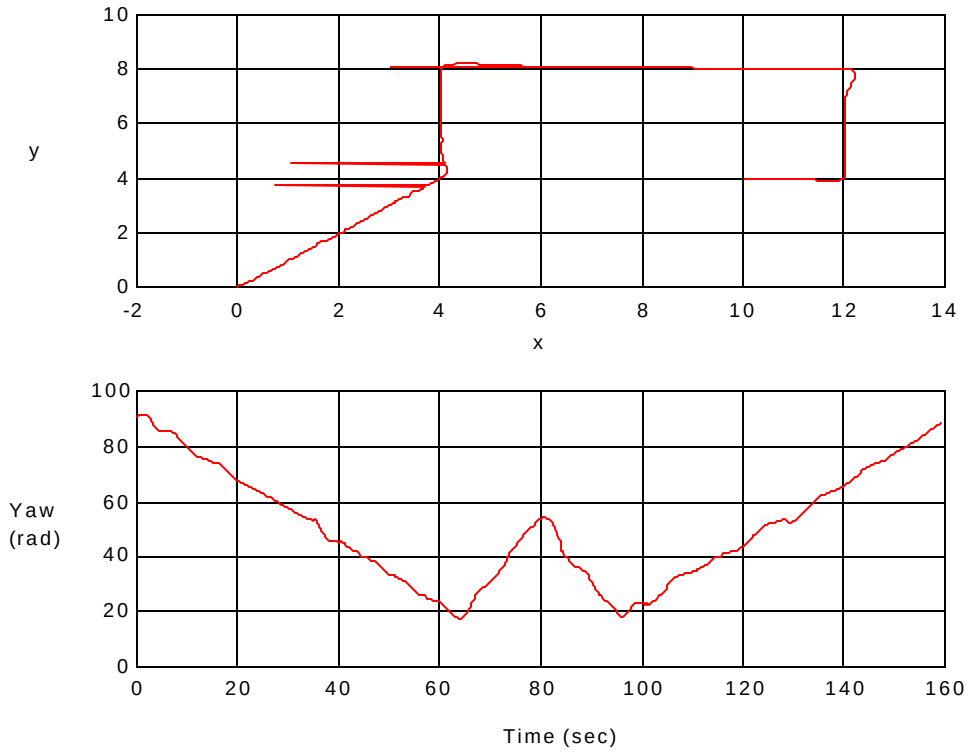

Figure 23. Experimental results.

Conclusion

We have described a novel six-wheeled “omnidirectional” autonomous robot platform based on the USU smart wheel mobility capability concept and a distributed, multiprocessor vetronics system. We also described the multiresolution behavior-generation strategy that was developed for these systems. A task decomposition approach using a grammar of primitive maneuvers is the basis for mission and path planning for the ODV robots, and a feedback linearization control strategy is used for path tracking. Simulations and experiments showed the effectiveness of the mission planning and control strategies. Ongoing and future research considers the mobility advantages offered by the ODV platform over more conventional systems and looks for ways to more tightly integrate the mission and path planner functions with the vehicle-level control systems to develop intelligent, reactive behaviors.

Acknowledgments

This research was conducted under the U.S. Army Tank-Automotive and Armaments Command (TACOM) Intelligent Mobility Program (agreement No. DAAE07-98-3-0023). The authors would like to thank the technical development team at USU's Center for Self-Organizing and Intelligent Systems for their efforts in implementing the ideas presented in this article. In particular, we acknowledge the work of Monte Frandsen, Gordon Olsen, and George Powell (vetronics and system engineering), Bob Gunderson and Carl Wood (electrical and mechanical), Shayne Rich, Morgan Davidson, and Vikas Bahl (mechanical and controls), and Paul Hepworth, Thomas Goodsell, and John Ogness (software and computer science). We are also grateful for the helpful comments of the reviewers.

References

[1] E. Poulson, J. Jacob, B. Gunderson, and B. Abbot, "Design of a robotic vehicle with self-contained intelligent wheels," *Proceedings of SPIE Conference on Robotic and Semi-Robotic Ground Vehicle Technology*, vol. 3366, Orlando, FL, pp. 68-73, April 15-16, 1998.

[2] *Webster's New Collegiate Dictionary*, Springfield, MA: G&C Merriam Co., 1981.

[3] C. Wood, M. Davidson, S. Rich, J. Keller, and R. Maxfield, "T2 omnidirectional vehicle mechanical design," *Proceedings of the SPIE Conference on Mobile Robots XIV*, Boston, MA, pp. 69-76, September 1999.

[4] M. Davidson and C. Wood, "Utah State University's T2 ODV mobility analysis," *Proceedings of SPIE Conference on Unmanned Ground Vehicle Technology*, vol. 4024-12, pp. 96-105, Orlando, FL, April, 2000.

[5] S. Rich, J. Keller, and C. Wood, "ODV mobility enhancement using active height control," *Proceedings of SPIE Conference on Unmanned Ground Vehicle Technology*, vol. 4024-16, pp. 137-145, Orlando, FL, April, 2000.

[6] H. McGowen, "Navy omnidirectional vehicle (ODV) development and technology transfer opportunities," Coastal Systems Station, Dahlgren Division, Naval Surface Warfare Division, unpublished report.

[7] M. Asama, M. Sato, H. Kaetsu, K. Ozaki, A. Matsumoto, and I. Endo, "Development of an omnidirectional mobile robot with 3 DOF decoupling drive mechanism," *Journal of the Robotics Society of Japan*, vol. 14, no. 2, pp. 95-100, 1997 (in Japanese).

[8] A. Mutambara and H. Durrant-Whyte, "Estimation and control for a modular wheeled mobile robot," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 1, pp. 35-46, January 2000.

[9] A. Rodic and M. Vukobratovic, "Contribution to integrated control synthesis of road vehicles," *IEEE Transactions on Control Systems Technology*, Vol. 7, No. 1, pp. 64-78, January 1999.

[10] R. Colyer and J. Economou, "Comparison of steering geometries for multi-wheeled vehicles by modeling and simulation," *Proceedings of 37th IEEE Conference on Decision and Control*, pp. 3131-3133, Tampa, FL, December 1998.

[11] J. Economou and R. Colyer, "Modeling of skid steering and fuzzy logic vehicle ground interaction," *Proceedings of 2000 American Control Conference*, pp. 100-104, Chicago, IL, June 2000.

[12] B.P. Ziegler, T.H. Cho, and J.W. Rozenblit, "A knowledge-based simulation environment for hierarchical flexible manufacturing," *IEEE Trans. On Systems, Man, and Cybernetics – Part A: Systems and Humans*, vol. 26, no. 1, pp. 81-90, January 1996.

[13] T. Cao and A.C. Sanderson, "AND/OR net representation for robotic task sequence planning," *IEEE Trans. on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, Vol. 28, No. 2, pp. 204-218, May 1998.

[14] R P. Bonasso, D Kortenkamp, D. P. Miller, M. G. Slack, "Experiences with an Architecture for Intelligent, Reactive Agents," In Michael Wooldridge, Jörg P. Müller, Milind Tambe (Eds.): *Intelligent Agents II, Agent Theories, Architectures, and Languages, IJCAI '95*, Workshop (ATAL), Montreal, Canada, August 19-20, 1995, Proceedings. Lecture Notes in Computer Science, Vol. 1037, pp. 187-202, Springer, 1996, ISBN3-540-60805-2.

[15] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan, and M.M. Veloso, "Xavier: Experience with a Layered Robot Architecture," *Agents '97*, 1997.

[16] P. Stone, M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," ARTIFICIAL INTELLIGENCE , 1999 110(2), pp 241—273.

[17] S. Russell, and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, NJ, USA, 1995.

[18] J.C. Latombe, *Robot Motion Planning*, Dordrecht, The Netherlands: Kluwer, 1991.

[19] R. E. Fayek, "Surface Modeling Using Hierarchical Topographic Triangular Meshes," Ph.D. Dissertation, University of Waterloo, Waterloo, Ontario, Canada, 1996.

[20] M. Tao M., A. Elssamadisy, N. Flann, and B. Abbott, "Optimal route re-planning for mobile robots: a massively parallel incremental A* algorithm," *Proceedings of IEEE International Conference on Robotics and Automation*, Albuquerque, NM, pp. 2727-2735, May 1997.

[21] Anthony Stentz, "The Focused D* Algorithm for Real-Time Re-planning." *International Joint Conference on Artificial Intelligence*, 1995, pp 1652-1659.

[22] M. Torrie, R. Koch, D. Cripps, "Ultra-maneuverable algorithms for terrain transitions," *Proceedings of SPIE Conference on Unmanned Ground Vehicle Technology*, Vol. 3693, pp. 66-78, Orlando, FL, April 7-9, 1998.

[23] K. Moore, M. Davidson, V. Bahl, S. Rich, and S. Jirgal, "Modeling and control of a six-wheeled autonomous robot," *Proceedings of 2000 American Control Conference*, pp. 1483-1490, Chicago, Illinois, June 2000.

[24] J. Wong, *Theory of Ground Vehicles*, New York: Wiley, 1978.

[25] H. Khalil, *Nonlinear Systems*, 2nd Edition, Upper Saddle River, NJ: Prentice-Hall, 1996.