January 28th, 2013

Humanoid League Technical Committee,

I hereby express the full commitment of team *Cyberlords La Salle*, which I formally represent, to participate in the 2013 edition of the *RoboCup World Championship* to take place in Eindhoven on June 24th – 30th.

I also express the full commitment of the team to provide at least one team member with sufficient knowledge of the rules to act as referee during the competition.

M. en C. LUIS FERNANDO LUPIÁN SÁNCHEZ
Profesor Investigador, Laboratorio de Robótica Móvil y Sistemas Automatizados
Facultad de Ingeniería, Universidad La Salle, México, D.F., MEXICO
+52(55)52789500 ext. 2286, lupianl@lci.ulsa.mx

Cyberlords RoboCup 2013 Humanoid KidSize Team Description Paper

Luis F Lupián, Diego Márquez, Omar Nelson, Francisco Lecumberri, and Iker Sanz

Mobile Robotics and Automated Systems Lab, Universidad La Salle, México
lupianl@lci.ulsa.mx http://www.cyberlordslasalle.org

Abstract. We describe the RoboCup KidSize humanoid robots to be used by team *Cyberlords La Salle* in the RoboCup 2013 competition to be held in Mexico City, Mexico. For this edition of the competition we will present *three* different robot architectures. One of them is based on the ROBONOVA-1 robot, and two on the KONDO KHR-3HV. Having different hardware architectures has forced our group to develop a software library called *libCyberlords*, which has undergone a major reorganization in preparation for 2013.

1 Introduction

Team *Cyberlords La Salle*, which is part of the *Mobile Robotics and Automated Systems Laboratory* at *Universidad La Salle* México, started its humanoid robot project in July 2008. The team debuted becoming champion at the *First RoboCup Mexican Open* in September 2008. Since then, the team has taken part in three *RoboCup World Championships*, two *RoboCup Latin American Opens* and two additional *RoboCup Mexican Opens*. The team has shown consistent improvement since its first *RoboCup World Championship*. In RoboCup 2009, team *Cyberlords La Salle* scored 2 goals and won one of its three matches. In RoboCup 2010, the team advanced to the second round and scored a total of six goals. In these two world championships the team participated in collaboration with the *Robotics and Artificial Vision Laboratory* of *Cinvestav* [1], [2]. In RoboCup 2011, where our team participated on its own, our top scorer Max scored seven goals and the team was just one goal away from reaching the quarter finals. At the *RoboCup Latin American Open* team *Cyberlords La Salle* became champion for two years in a row, 2010 and 2011. In the *RoboCup Latin American Open 2011*, which took place in Bogotá, Colombia, our top scorer Max broke its own record by scoring nine goals in four matches (Fig. 1).

During 2012 our team worked in collaboration with team *Falconbots Tec San Martín* under the name *Cyberlords + Falconbots* [3]. Under this collaboration, our team conquered the Mexican Championship in April 2012 by winning three out of four matches and tying the other one at the *RoboCup Mexican Open 2012* (Fig. 2). During RoboCup 2012 in Mexico City, our team advanced in 1st place of its first-round group after defeating the Indonesian team by 3 to 1. It

Fig. 1. Max at the *RoboCup Latin American Open*, Bogotá, Colombia, October 2011

must be noted that for RoboCup 2012 our team played with two DARWIN-OP robots, one of them provided by team *Falconbots Tec San Martín* and the other one by Prof. Jacky Baltes, from the University of Manitoba, to whom we owe our gratitude.

Fig. 2. Max shooting for the final goal at the *RoboCup Mexican Open*, April 2012

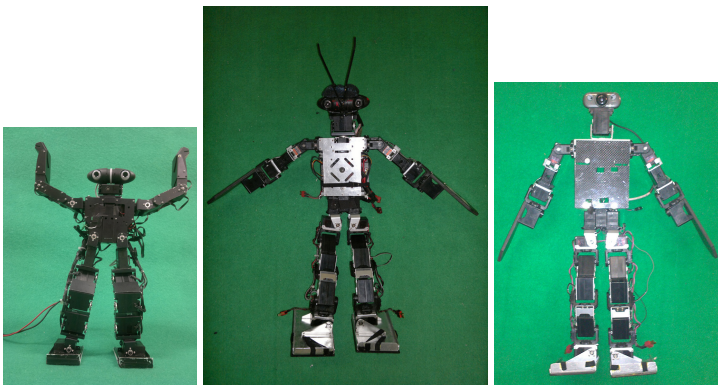
2 Hardware Architectures

One of our main objectives in RoboCup 2012 was to demonstrate the portability and interoperability of our humanoid robot software library *libCyberlords*. For that edition of the Wolrd Cup we competed as a joint team composed of a ROBONOVA-1 based robot, two KONDO KHR-3HV based robots and two DARWIN-OP. This gave us the opportunity to test the library on different hardware architectures. In the case of both DARWIN-OP robots only the vision and decision sub-systems of *libCyberlords* were used, since our colleagues from *Falconbots Tec San Martín* decided to develop their own locomotion system. Nonetheless, this was sufficient to prove our point. Both DARWIN-OP robots on our team outperformed our top scorer Max despite the fact that *libCyberlords* was not originally designed with the DARWIN-OP architecture in mind. The two DARWIN-OP robots in our team scored a total of three goals in one of our matches at RoboCup 2012 (Fig. 3).

Fig. 3. Team *Cyberlords + Falconbots* scoring at *RoboCup 2012*, Mexico City

This year we intend to participate using only our three humanoid robots (shown in Fig. 4), whose hardware architecture is described in full in the accompanying specification sheets.

One major change in our newest architecture (T3) is the fact we are using two Gumstix Overo Fire units, one for the vision system and the other for locomotion, perception and decision, plus an RCB4 for servomotor interface. The three computing units fit perfectly into a small-sized box which can easily be removed from the body of the robot for maintenance (see Fig. 5).



Architecture T1 Architecture T2 Architecture T3

Fig. 4. Three hardware architectures for team *Cyberlords La Salle* in 2013

3 Task Concurrency in *libCyberlords*

High performance concurrency is an essential feature of a highly demanding mobile robotics application, especially in the case where computing power is restricted by the application, such as in the case of a small humanoid robot. We

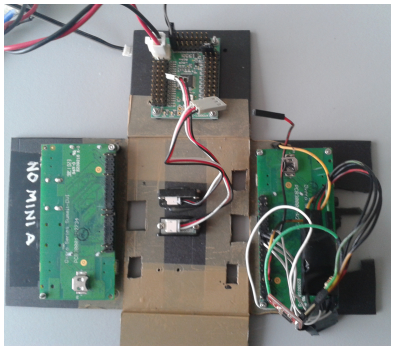


Fig. 5. Processing units for architecture T3

decided to implement concurrency in *libCyberlords* in the form of cooperative multitasking. This way can write an application-specific scheduler which lets us have precise control over how much computing power is dedicated to each task.

Cooperative multitasking is implemented in *libCyberlords* as an abstract base class called Cyberlords::Task. Listing 1 shows the essential elements of this class. The most important ones are the pure virtual methods Task::execute_slice_ and Task::restart_. The method Task::execute_slice_ must be implemented in each concrete descendant to specify what "slice" of code has to execute next. This method returns a Task::Status to let the scheduler know whether the task has finished or another slice of code is waiting to be executed. Task::Status is one of Error, Busy or Done. The virtual method Task::restart_ lets each concrete descendant specify how to initialize the internal state of the task right before it is started by the scheduler.

Listing 1. class Cyberlords::Task

```
1   class Task
2   {
3   protected:
4       virtual Status execute_slice_(void) = 0;
5       virtual void restart_(void) = 0;
6   public:
7       Task(void);
8       Status execute_slice(void);
9       Status status(void);
10  };
```

4 Hardware Abstraction

Abstracting hardware in a mobile robot application helps isolate the implementation details of the lower software levels from the decision layer. In this way the higher decision layer can be designed uniformly for all humanoid robots regardless of their hardware differences. Hardware components in a mobile robot can be classified as "perception hardware" and "locomotion hardware". In *libCyberlords*, a different approach was taken to implement abstraction on each of these two classes of hardware components.

4.1 Locomotion Abstraction

Within *libCyberlords* the basis for locomotion abstraction is implemented in class Cyberlords::MotionBehavior. Listing 2 shows the essential components of this class, which are the pure virtual methods MotionBehavior::play and MotionBehavior::stop. All concrete descendants of this class must implement these two methods in order to specify each atomic motion behavior. The list of motion behaviors is application-dependent. For the case of playing soccer a total of thirty motion behaviors are to be specified. Among them: IniLftStepFwd, IntLftStepFwd, FinLftStepFwd, IniLftStepRev, LatStepLft, KickLft, FaceUpStandUp, Squat, SitDown, DiveLft.

Listing 2. class Cyberlords::MotionBehavior

```
1     class MotionBehavior: public Task
2     {
3         virtual int play(void) = 0;
4         virtual int stop(void) = 0;
5     };
```

Another important component of locomotion abstraction is the class Cyberlords::MotionBehaviorEngine, where the developer must register each of the motion behaviors specified for the hardware architecture. A simplified version of this class is shown in Listing 3.

Listing 3. class Cyberlords::MotionBehaviorEngine

```
1     class MotionBehaviorEngine: public Task
2     {
3     protected:
4         void register_motion_behavior
5         (
6             MotionBehavior::Enumerator id,
7             MotionBehavior &motion_behavior
8         );
9     public:
10        int play(MotionBehavior::Enumerator id);
11        int stop(void);
12    };
```

4.2 Perception Abstraction

The abstraction of perception hardware components is implemented for each architecture by defining a set of abstract questions such as: HaveIFallen, WhereIsBallRelativeToMe, WhereAmIRelativeToField, WhereIsRivalGoalRelativeToMe. These questions are abstract in the sense that the answer does not represent a numeric value but rather an abstract meaning. Listing 4 shows a sample abstract question for which the possible abstract answers are Not, AlmostFront, Front, AlmostBack, Back, AlmostLeft, Left, AlmostRight, Right. In this way, the decision layer is isolated from the implementation details that differentiate each hardware architecture.

Listing 4. class Cyberlords::HaveIFallen

```
1     class HaveIFallen: public Task
2     {
3     public:
4         HaveIFallen(void);
5         HowFallen answer(void);
6     };
```

5 Decision Layer

Each abstract question is associated with an abstract boolean condition at the decision layer. This is done using the abstract base class shown in Listing 5.

Listing 5. class Cyberlords::Condition

```
1   class Condition
2   {
3   public:
4       virtual ~Condition(){};
5       virtual bool eval(void) = 0;
6       virtual Condition* clone(void) const = 0;
7   };
```

Listing 6 shows a sample abstract condition which indicates whether the robot fell or not.

Listing 6. class Cyberlords::ConditionRobotFallen

```
1   class ConditionRobotFallen: public Condition
2   {
3   private:
4       HaveIFallen &have_i_fallen;
5   public:
6       ConditionRobotFallen(HaveIFallen &have_i_fallen_);
7       bool eval(void);
8       Condition* clone(void) const;
9   };
10  #define CL_ROBOT_FALLEN ConditionRobotFallen(have_i_fallen)
```

At the decision layer *libCyberlords* implements two abstract descendants of Cyberlords::Task: Cyberlords::State and Cyberlords::StateMachine, see Fig. 6. These two classes, in combination with Cyberlords::Condition are the basis for constructing decision behaviors in *libCyberlords* based applications. The fact that Cyberlords::StateMachine is a descendant of Cyberlords::State is how a hierarchy of state machines can be built to specify complex behaviors for simpler ones.

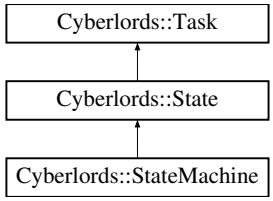


Fig. 6. Task-State-StateMachine hierarchy in *libCyberlords*

One powerful feature of *libCyberlords* is a metalanguage that is used to construct state machines using simple transition tables. The class Cyberlords::Transition is essentially a triplet composed of an origin state, a destination state and a Cyberlords::Condition. Listing 7 shows a sample code written using the *State Machine Metalanguage*, which represents a simple behavior for getting up after a fall. This state machine has only two states: INITIAL and GETUP. As long as the condition ROBOTFALLEN is true the state machine will transition to GETUP

otherwise it exits. More complex behaviors can be constructed by defining simple state machines and then composing them into a hierarchy. For example, Listing 8 shows a state machine with five states, out of which four are state machines themselves: WalkToBall, RunToBall, AimAtGoal, KickAtGoal. Listing 8 also shows more complex high level conditions, which are another important feature of *libCyberlords*.

Listing 7. GetUp State Machine

```
1     //   Origin     Destination  Condition
2     // +--------+-----------+-------------------+
3     RT( INITIAL, GETUP,       ROBOT_FALLEN        );
4     RT( INITIAL, EXIT,        DEFAULT_TRANSITION  );
5     // +--------+-----------+-------------------+
6     RT( GETUP,   GETUP,       ROBOT_FALLEN        );
7     RT( GETUP,   EXIT,        DEFAULT_TRANSITION  );
8     // +--------+-----------+-------------------+
```

Listing 8. Walk and get up

```
1     //   Origin       Destination  Condition
2     // +-----------+-----------+-------------------------------------------------+
3     RT( INITIAL,    WALKTOBALL, PERCEIVE_BALL && BALL_D(Near)                    );
4     RT( INITIAL,    RUNTOBALL,  PERCEIVE_BALL && BALL_D(Far)                     );
5     RT( INITIAL,    AIMATGOAL,  PERCEIVE_BALL && BALL_D(OnFeet) && !GOAL_O(Front));
6     RT( INITIAL,    KICKATGOAL, PERCEIVE_BALL && BALL_D(OnFeet) && GOAL_O(Front) );
7     RT( INITIAL,    INITIAL,    DEFAULT_TRANSITION                               );
8     // +-----------+-----------+-------------------------------------------------+
9     RT( WALKTOBALL, RUNTOBALL,  PERCEIVE_BALL && BALL_D(Far)                     );
10    RT( WALKTOBALL, AIMATGOAL,  PERCEIVE_BALL && BALL_D(OnFeet) && !GOAL_O(Front));
11    RT( WALKTOBALL, KICKATGOAL, PERCEIVE_BALL && BALL_D(OnFeet) && GOAL_O(Front) );
12    RT( WALKTOBALL, INITIAL,    DEFAULT_TRANSITION                               );
13    // +-----------+-----------+-------------------------------------------------+
14    RT( RUNTOBALL,  WALKTOBALL, PERCEIVE_BALL && BALL_D(Near)                    );
15    RT( RUNTOBALL,  AIMATGOAL,  PERCEIVE_BALL && BALL_D(OnFeet) && !GOAL_O(Front));
16    RT( RUNTOBALL,  KICKATGOAL, PERCEIVE_BALL && BALL_D(OnFeet) && GOAL_O(Front) );
17    RT( RUNTOBALL,  INITIAL,    DEFAULT_TRANSITION                               );
18    // +-----------+-----------+-------------------------------------------------+
19    RT( AIMATGOAL,  KICKATGOAL, PERCEIVE_BALL && BALL_D(OnFeet) && GOAL_O(Front) );
20    RT( AIMATGOAL,  INITIAL,    DEFAULT_TRANSITION                               );
21    // +-----------+-----------+-------------------------------------------------+
22    RT( KICKATGOAL, INITIAL,    DEFAULT_TRANSITION                               );
23    // + ----------+-----------+-------------------------------------------------+
```

An important thing to notice in the previous two listings is the fact that all decision behaviors are specified independently of any implementation details from the lower layers. This means that these state machines are directly portable among different hardware architectures without modification as long as the same motion behaviors and abstract questions are implemented. For the case of soccer-playing robots, this means that as long as a humanoid robot has the hardware features for playing soccer it should be possible for it to share *libCyberlords* decision behaviors with other soccer-playing humanoid robots regardless of their hardware differences.

6 Conclusion and Future Work

We have described the hardware and software features of the humanoid robot architectures to be used by team *Cyberlords La Salle* in the RoboCup 2013 world championship. Our current work is concentrated in reorganizing and consolidating the humanoid robot library we introduced in mid 2011 (*libCyberlords*). We are in the process of organizing the software project under cmake so that it can easily be compiled on different hardware platforms and development tools. Using

cmake we will also be able to implement automated testing so as to ensure the reliability of *libCyberlords*. The project will be released as a git repository to facilitate sharing of code and concurrent development. It is our hope to be able to release the first public version of *libCyberlords* before RoboCup 2013.

Another focus area for our current efforts is turning the *State Machine Metalanguage*, which is already present in *libCyberlords*, into a complete language by developing a language-specific compiler which will simplify even further the process of specifying states and state machines for the decision layer of *libCyberlords* based applications.

Acknowledgements

This project is supported in part by *CONACyT*, the *Mexican Robotics Federation* and the *Red Nacional de Robótica y Mecatrónica*.

Team Members

Team *Cyberlords La Salle* for 2013 will be integrated by at least the following people:

- Team leader: Prof. Luis F. Lupián.
- Team members: Diego Márquez, Iker Sanz, Omar Nelson, Francisco Lecumberri, Guillermo Oviedo, Fernando Chávez.

References

1. Lupián, L.F., Romay, A.I., Monroy, P., Espínola, A.F., Cisneros, R., Benítez, F.: Cyberlords RoboCup 2009 Humanoid KidSize team description paper. In: RoboCup World Championship, Graz, Austria, RoboCup Federation (July 2009)
2. Lupián, L.F., Romay, A., Espínola, A., Cisneros, R., Ibarra, J.M., Gutiérrez, D., Hunter, M., del Valle, C., de la Loza, K.: Cyberlords RoboCup 2010 Humanoid KidSize team description paper. In: RoboCup World Championship, Singapore, RoboCup Federation (June 2010)
3. Lupián, L.F., Romay, A., Espínola, A., Márquez, D., Reyes, D.M.: Cyberlords+Falconbots RoboCup 2012 Humanoid KidSize team description paper. In: RoboCup World Championship, Mexico City, Mexico, RoboCup Federation (June 2012)
4. Lupián, L.F., Romay, A., Espínola, A., Ramírez, E.: Cyberlords RoboCup 2011 Humanoid KidSize team description paper. In: RoboCup World Championship, Istanbul, Turkey, RoboCup Federation (July 2011)
5. Lupián, L.F., Romay, A., Espínola, A.: Vision Based Localization of Humanoid Robots by Inverse Pose-Estimation Using a Small Set of Fixed Landmark Features. In: Robotic Symposium, IEEE Latin American, Bogotá, Colombia (October 2011)
6. Romay Tovar, A.I.: Visión computacional para problemas de estimación de pose. Master's thesis, Universidad La Salle, D.F., México (2011)
7. Espínola Auada, A.F.: Sistema de visión de un robot humanoide en un ambiente semi estructurado. Master's thesis, Universidad Nacional Autónoma de México, D.F., México (2011)