Jeah
control systems

April 2nd, 2002

End-to-end Prototype Report

Laboratory #08

Revision 2

Prepared by: Abraham C. Lleva & Hao Liang

Jeah Control Systems

CS 340, Group 25

Table of Contents

2 Preface

Jeah Control Systems (Group 25) and Serendipitous Solutions (Group 26) made a combined effort to develop and demonstrate an intermediate prototype for the mobile robot system. The purpose of this document is to record the goals, schedule, testing performance, and actual implementation used for the prototype.

3 Prototype Goals

Both groups were required to modularize and develop their parts of the prototype system separately. Sharing the same communication protocol and the utilization of object-oriented design, the general strategy would be to reconcile the input/output parameters between the system modules and after implementation, bridge the two pieces through these parameters.

After both pieces were completed, they were to be immediately brought together and run through an integration test. Success would be achieved once the two halves of the prototype were working in tandem.

Once the nature of this joint development plan was agreed upon by both parties such that both companies would require the least amount of interaction with each other during the design/implementation process, we separated into our original groups. The following sections outline first the overall system goal for the project and then the basic goals/milestones set by our group for our half of the prototype (the onboard software mechanism).

3.1 System Goal

The robot onboard software would first be required to connect to both Group 26's interface system as well as the actual robot. Then, integration with Group 26's interface was expected to provide a user the ability to perform the following functions.

- Connect to robot
- Move forward
- Stop
- Turn Camera
- Simultaneous control of both robot and camera

3.2 Onboard Software Goals

The following table describes the goals we set for our development process of the onboard software as well as our reasons for doing so.

Goal	Explanation
Create a Socket	It is necessary to create a socket successfully so that communication between the onboard software and the control station can be established in the future.
Create a Listener	An intermediate goal that aims at giving the server ability to receive data sent from the control station.
Map Control Station Interface	After creating a socket and a listener, the buttons on

Buttons to Onboard Software Functions.	the control station interface should be mapped correctly to a specific function of the onboard software.
Control Robot Motion	Correctly implement onboard software functions so that the user can move the robot forward at a predefined speed and stop the robot using the control station interface.
Control Camera Motion	Correctly implement onboard software functions so that the user can pan the camera to the left, and pan the camera to the right using the control station interface.
Implement Thread Management	Correctly manage threads so that the camera and the robot can function at the same time.

•Description Table: Description of Onboard Software Goals

4 Schedule

There was a one-week deadline enforced upon this project. Therefore, it was very important that a schedule was created for the organization of our work. Our group set out the following schedule plan for the creation of our onboard software system module. This section also comments on the actual success/completion of each individually scheduled goal.

4.1 Detailed Milestone Schedule

The following are the milestones we set for our development process. Each separate milestone was assigned to a group member(s), given a certain deadline date for its completion. The sequence decided upon was such that milestones be depended on by other milestones were scheduled earlier and milestones that could be accomplished separately were assigned in parallel—this was looked upon as the most efficient, logical method of scheduling. Other concerns such as location, resources, etc. were decided by the member responsible and are not included in this report for the sake of conciseness.

Milestone	Explanation	Developer	Deadline
Server Connection	A correct implementation of the server that allows the test tool to send a move command to be sent by a user and received by the onboard software.	Emmanuel Smadja	Sunday, March 24
Movement Response	An untested implementation of the robot move forward function that would adequately translate the user command and control the robot.	Jit Sarkar	Sunday, March 24
Camera Response	An untested implementation of the camera left/right turn function that would adequately translate the user command and control the robot camera.	Jit Sarkar	Sunday, March 24
Simulator Response	A successful testing of both robot movement and camera turning implementation with the Nomadic Simulator.	Jit Sarkar	Tuesday, March 26
Robot Response	A successful testing of the robot forward movement and camera turn implementation with the Nomadic Scout.	Jit Sarkar	Tuesday, March 26
Module Integration	A working integration of the server and command implementations with the Nomadic Scout.	Emmanuel Smadja	Tuesday, March 26

Final Integration	A successful integration and testing of both groups' modules that allows a user to send commands for forward movement and camera turning from the interface and receive an expected response from the robot.	Jit Sarkar, Emmanuel Smadja, Developers of Group 26	Tuesday, March 26

•**Description Table:** Schedule of Milestones and Assignment of Responsibility

5 Testing Report

The following explains the actual schedule process that was carried out and the result of the many tests that were applied. We are happy to indicate that the project was an overall success and that all goals were completed.

5.1 Post-Development Testing Report

The following table includes information concerning the success/failure of our schedule and any problems encountered throughout the development process. The problem that was of most concern during the development process was being refused access to the robot the night of final testing—it greatly increased the risk of not being able to test/complete the project before deadline (but fortunately did not result as such).

Milestone	Problems Encountered	Developers Present	Date Completed
Server Connection	No major problems encountered.	Emmanuel Smadja	Sunday, March 24
Movement Response	No major problems encountered.	Jit Sarkar	Sunday, March 24
Camera Response	No major problems encountered.	Jit Sarkar	Sunday, March 24
Simulator Response	No major problems encountered.	Jit Sarkar	Tuesday, March 26
Robot Response	Unable to use robot Tuesday night due to CS216 labs.	Jit Sarkar	Wednesday, March 27
Module Integration	Unable to use robot Tuesday night due to CS216 labs.	Emmanuel Smadja	Wednesday, March 27
Final Integration With Client Module	Unable to use robot Tuesday night due to CS216 labs. Initial difficulty connecting with robot interface and finding open communication ports.	Jit Sarkar, Emmanuel Smadja, Sean Jellish, Tiffany Lane, Jennifer Henderson	Wednesday, March 27

•**Description Table:** Schedule Problems and Completions

6 Source Code

The final implementation of our part of the prototype is included in the following sections in C++.

6.1 Main.cpp

```
#include "Robot_module.h"
#include "Rmodule.h"
#include <pthread.h>

void *Input_Loop(void *argument);
```

```
int main (int argc, char *argv[])         {

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    int status;

    pthread_t Input_thread;

    Robot_Module Robot(atoi(argv[1]), argv[2], atoi(argv[3]));

    Rmodule Receive(&Robot);

    cout << endl << "Robot Created" << endl;

    pthread_create( &Input_thread, &attr, Input_Loop, (void *) &Robot);

    Receive.net_in();


    pthread_join(Input_thread, (void **)&status);

    exit(0);
}


void *Input_Loop(void *argument) {

    Robot_Module *Robot = (Robot_Module *) argument;

    char input;

    float vel = 0;
    float velChange = 10;

    while ((cin >> input))       {

       switch (tolower(input))   {
              case 'q':
                       cerr << endl << "quitting..." << endl;
                       exit(0);
                       pthread_exit((void *) 0);
                       break;

              case '[':
                       cerr << endl << "reducing speed.." << endl;
                       Robot->SetVel(vel += velChange);
                       break;

              case ']':
                       cerr << endl << "increasing speed.." << endl;
                       Robot->SetVel(vel -= velChange);
                       break;

              case 's':
                       vel = 0;
                       Robot->Stop();
                       break;

              default:
                       break;
       }

    }


}
```

6.2 Makefile

```
#############################################################################
#                                                                           #
#                   CS 340 EXAMPLE MAKEFILE                                 #
#                                                                           #
#############################################################################
#
# Everything following a '#' is a comment in a makefile
#

# Here we define a macro for the compiler name.  This is just in
# case we wish to change The compiler at some point in the future

CC     =      g++


# This will be the list of all the flags that have to do with
# compiling a C/C++ program.  Other possible flags might include
# -g for debugging, or -O for optimization

CFLAGS        =         -fpermissive -I/usr/local/nomad/client

# This is the list of all desired linker flags

LFLAGS        =         -lpthread

# The libraries to link into the binaries

LIBS          =

# just a macro for removing files

RM     =      /bin/rm -f



# The OBJECTS macro is a list of all of the objects that will be
# compiled and linked into the binaries.  The '\' at the end of a
# line indicates that the line should continue on the next text line
# without a carriage return

OBJECTS                 =         Robot_module.o Rmodule.o

# Likewise, this is a list of all of the BINARIES which this makefile
# may generate.


BINARIES      =         myreal mysim

#############################################################################

# The rest of this makefile consists of make rules.  This
# one called 'all' tells the make program that 'all'
# depends on everything in the BINARIES macro to be
# up-to-date

all:   $(BINARIES)


# Likewise, the clean dependent itself depends on nothing
# (i.e. the blank dependency list), but that when the
# rule is executed, it should remove all binaries and
# objects

clean:
    $(RM) $(BINARIES) $(OBJECTS)


# The following rule is an example of a rule which creates an object
# file.  The rule says that connect_sim.o is out of date whenever its
```

```
# time stamp does not match that of connect_sim.c.  If connect_sim.c
# has been edited since the last time connect_sim.o was created, then
# the rule should be executed.


Robot_module.o:        Robot_module.cpp Robot_module.h
    $(CC) $(CFLAGS) Robot_module.cpp -c -o Robot_module.o

Rmodule.o:    Rmodule.cpp Rmodule.h
    $(CC) $(CFLAGS) Rmodule.cpp -c -o Rmodule.o


# This likewise indicates that connect_sim depends on all of the listed
# objects to be up to date.  If not, it should executed its own rule.
# NOTE that if any of the objects have rules them selves which indicate
# that they are not up to date, then those rules most first execute and
# make will do so for the user.

mysim: main.cpp $(OBJECTS)
    $(CC) $(CFLAGS) main.cpp $(OBJECTS) Nclient.o -o mysim  $(LFLAGS) $(LIBS)


myreal:      main.cpp $(OBJECTS)
    $(CC) $(CFLAGS) main.cpp $(OBJECTS) Ndirect.o -o myreal  $(LFLAGS) $(LIBS)
```

6.3 RModule.h

```
#ifndef _RMODULE_H_
#define _RMODULE_H_

#include <fstream>
#include <queue>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <netdb.h>
#include <string>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/param.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <iostream> // needed for the prototype

#include "Robot_module.h"

// class Rmodule
// Creates The reception module (object) which
// will be used to translate and prioritize messages
// that the communication protocol received from the
// user interface.


class Rmodule
{
public:
    // constructor
    Rmodule(Robot_Module *robot);

    // destructor
    //~Rmodule();

    //inspectors

    // mutators
```

```
    //others
    // Receives message from the interface
    int net_in();
    // send a regular command to the robot
    void send_bot_cmd(int sock, char *message);
    // send an urgent command to the robot
    //void send_priority(int cmd);
    // defines whether the command is urgent or not
    //bool prioritize(int cmd);
    // sends event ("message sent", "message received",...) to log
    //void send2log(ostream &sout);
    // translates int message into robot commands
    //string translate(int message);

private:
//  queue Rqueue;
//  friend class CommProtocol;
    Robot_Module *Robot;
};

#endif
```

6.4 RModule.cpp

```
// RModule.cpp
#include "Rmodule.h"
// constructor
Rmodule::Rmodule(Robot_Module *robot){

    Robot = robot;

}

// destructor
// Rmodule::~Rmodule{}

//inspectors

// mutators

//others
// gets message from interface and sends a regular command to the robot
void Rmodule::send_bot_cmd(int sock, char *message){
  // process communication received over TCP port
  // handle messages
  int MaxVel = 100;
  cout << "the message passed is: " << message[0] << endl;


  int len = strlen(message);

  if (((int) message[0] == 1) && ((int) message[1] == 1)){
    cout << "move robot command received" << endl;
    Robot->SetVel(MaxVel);
  }

  //else if (){
  //  cout << "pan camera command received" << endl;
  //  robot->camera.pan();
  //}

  else if (((int) message[0] == 1) && ((int) message[1] == 2)){
    cout << "stop command received" << endl;
    Robot->Stop();
  }
  //send(sock, message, len, 0);

}
int Rmodule::net_in()
```

```
{

    const int MaxBuffer  = 1024;
    const int ServerPort = 2220; // used to be 8991
    char                        message[MaxBuffer + 1];
    int                          Socket, AcceptedSocket, len;

    struct sockaddr_in server;
    struct hostent              *HostInfo;


    //
    // Open a socket and start listening for connections
    //

    memset( &server, 0, sizeof( struct sockaddr_in ) );
    gethostname( message, MaxBuffer );
    HostInfo = gethostbyname( message );

    if ( HostInfo == NULL )
    {
       perror( "could not get hostname" );
       exit( 1 );
    }

    server.sin_family = HostInfo->h_addrtype;
    server.sin_port = htons( ServerPort );
    Socket = socket( AF_INET, SOCK_STREAM, 0 );

    if ( Socket < 0 ) {
       perror( "could not create socket" );
       exit( 1 );
    }

    if ( bind( Socket, ( struct sockaddr * ) &server, sizeof( struct sockaddr_in ) )
       < 0 ) {
       perror( "could not bind socket" );
       exit( 1 );
    }

    listen( Socket, 0 );

    //
    // Handle requests one at a time
    //

    while ( 1 ) {
       int size = 3;
       AcceptedSocket = accept( Socket, NULL, NULL );

       if ( AcceptedSocket < 0 )
              continue;

       memset( message, 0, MaxBuffer );
       while ( ( len = read( AcceptedSocket, message, MaxBuffer ) ) > 0 ) {
              send_bot_cmd( AcceptedSocket, message );
              memset( message, 0, MaxBuffer );
              send(AcceptedSocket, "YO!", size, 0);
       }

       close( AcceptedSocket );
    }

    close( Socket );   // This should never be reached
    return 0;
}

// send an urgent command to the robot
// void Rmodule::send_priority(int cmd){}
// defines whether the command is urgent or not
// bool Rmodule::prioritize(int cmd){}
```

```
// sends event ("message sent", "message received",...) to log
// void Rmodule::send2log(ostream &sout){}
// translates int message into robot commands
// string Rmodule::translate(int message){}
```

6.1 RobotModule.h

```
#ifndef _ROBOT_MODULE_H_
#define _ROBOT_MODULE_H_

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string>
#include <iostream>

#include <pthread.h>
#include <sched.h>
#include <semaphore.h>
#include "Nclient.h"
//#include "CommProtocol.h"
//#include "Camera_module.h"
//#include "Sensors_module.h"

using namespace std;

// Robot Module Class, to provide interface with robot hardware
class Robot_Module     {


    public:
       //_______________
       // Constructor
       Robot_Module(int robot_ID, string address, int port_baud);
                       // "robot_ID", the number of the robot to connect to
                       // "address", the location of the robot, either a network
       name or a local serial-port
                       // "port", the port number for the network name, or the baud-
       rate for local serial


       //_______________
       // Destructor
       ~Robot_Module();


       //____________________
       // Public members

       // Public camera and sensor objects for direct manipulation:
       // e.g. Robot.Camera.SetPanAbs(20);
//     Camera_Module Camera;
//     Sensors_Module Sensors;


       //_______________
       // Inspectors

       // Returns the current velocity of the robot (-100% to +100%)
       float GetVel();

       // Returns the current acceleration of the robot (-100% to +100%)
//     float GetAccel();

       // Returns the current rate of turning of the robot (-100% to +100%)
//     float GetTurn();

       // Returns the user defined maximum velocity (0 to 100%)
//     float GetMaxVel();
```

```
        // Returns the user defined Maximum acceleration (0 to 100%)
//      float GetMaxAccel();

        // Returns the user defined Maximum Rate of Turning (0 to 100%)
//      float GetMaxTurn();

        // Returns the Address of the robot (network or local port)
//      string GetNetName();

        // Returns the Port Number(for network address) or Baudrate (for local port)
//      int GetNetPort();

        // Returns the ID number of the robot in use
//      int GetNetID();

        // Returns the maximum velocity of the robot hardware
//      float GetRobot_MaxVel();

        // Returns the Maximum acceleration of the robot hardware
//      float GetRobot_MaxAccel();

        // Returns the Maximum rate of turning of the robot hardware
        // Fictitious: robot is a differential motion platform
//      float GetRobot_MaxTurn();

        // Returns the timeout period set for the robot hardware
//      float GetTimeout();

        // Returns the Alert Response time interval
//      float GetAlertResponseLimit();


        //____________
        // Mutators

        // Sets a new velocity for the robot
        int SetVel(float new_Vel);          // Returns 1 if successful, otherwise 0
                                    // "new_Vel", the value of the new velocity (-
        max_vel to +max_vel)

        // Sets a new acceleration for the robot
//      int SetAccel(float new_Accel);      // Returns 1 if successful, otherwise 0
                                    // "new_Accel", the value of the new velocity (-
        max_accel to +max_accel)

        // Sets a new turning rate for the robot
//      int SetTurn(float new_Turn);        // Returns 1 if successful, otherwise 0
                                    // "new_Turn",     the value of the new turning
        rate (-max_turn to +max_turn)

        // Stops all robot motion
        int Stop();      // Returns 1 if successful, otherwise 0

        // Sets a new maximum allowable velocity for the robot
//      int SetMaxVel(float new_max);       // Returns 1 if successful, otherwise 0
                        // "new_max", the value of the new maximum (0 to +100%)

        // Sets a new maximum allowable Acceleration
//      int SetMaxAccel(float new_max);     // Returns 1 if successful, otherwise 0
                        // "new_max", the value of the new maximum (0 to +100%)

        // Sets a new maximum allowable rate of turning
//      int SetMaxTurn(float new_max);      // Returns 1 if successful, otherwise 0
                        // "new_max", the value of the new maximum (-100% to +100%)

        // Sets a new timeout period for the robot
        int SetTimeout(float new_time);     // Returns 1 if successful, otherwise 0
        // "new_time", the value of the new timeout period (0s to Robot_max_timeout)

        // Triggers an alert (e.g. because a bump switch was tripped):
```

```
      // Stops all robot motion, sends message to communications system and
      sets //the alert flag
//    int SetAlert(); // Returns 1 if successful, otherwise 0

//  Sets the alert acknowledged flag, indicating that the user is aware of the
      //problem
//    int AcknowledgeAlert();   // Returns 1 if successful, otherwise 0

      // Clears the alert flag, allowing normal operation to continue
//    int ClearAlert();         // Returns 1 if successful, otherwise 0

      // Sets a new Alert Response time interval
//    int SetAlertResponseLimit(float new_limit);
// Returns 1 if successful, otherwise 0
// "new_limit", the new time window in which to address an alert issue



   private:

      //____________________
      // Private members

      // Robot Hardware specific constants
      // the maximum allowed velocity, as defined by the robot hardware
      const int  Robot_max_vel = 400;

      // the maximum allowed accelerations, as defined by the robot hardware
      const int Robot_max_accel = 390;

      // the maximum allowed rate of turning for the robot, fictitious
      // robot has no inherent steering system, it is a differential platform
      const int Robot_max_turn = 100;


      // the frequency at which commands are sent to the robot hardware
      const int Refresh_rate = 100; //    in miliseconds

      // the maximum allowed timeout period, as defined by the robot hardware
      const int Robot_max_timeout = 255;

      // pointer to the communications system object
//    Comm_Protocol *Server;

      // defines the robots stopped mode
//    bool Stopped;

      // the address of the robot, either a network location, or a local port
      string Net_Name;

      // the port number of the robot(for network), or baud rate(for local port)
      int Net_Port;

      // the ID number of the robot being used
      int Net_ID;

      // the alert flag, set when a problem occurs
//    bool Alert;

      // the alert acknowledged flag is set to indicat that the operator is aware
      of the problem
//    bool Alert_Acknowledged;

      // the time period after acknowledgment is received in which robot motion is
      permitted
      // in order to correct the alert
//    float Alert_Response_Limit;

      // the currently set velocity of the robot (-max_vel to +max_vel)
      float current_vel;
```

```
        // the currently set acceleration of the robot (-max_accel to +max_accel)
        float current_accel;

        // the currently set rate of turning for the robot (-max_turn to +max_turn)
//      float current_turn;

        // the user defined maximum velocitiy of the robot (0 to 100%)
        float max_vel;

        // the user defined maximum accleration of the robot (0 to 100%)
        float max_accel;

        // the user defined maximum velocitiy of the robot (0 to 100%)
//      float max_turn;

        // the timeout period for the robot, the maximum allowed gap between commands
        issued
        // to the hardware, before the robot stops itself (0s to Robot_max_timeout)
        float Robot_timeout;


        //     Mutexes
        struct timespec delay;

        pthread_attr_t attr;

        pthread_t Control_thread;

        pthread_mutex_t Control_Lock;

        float left_wheel;
        float right_wheel;


        //_________________
        // Private member functions

        // private control loop thread that resends commands to the robot hardware
        // every (Refresh_rate)
        void *Control_Loop(void *not_used);


};

#endif
```

6.2 RobotModule.cpp

```
#include "Robot_module.h"

Robot_Module::Robot_Module(int robot_ID, string address, int port_baud)         {

    Net_Name = address;
    Net_ID = robot_ID;
    Net_Port = port_baud;

    strcpy ( SERVER_MACHINE_NAME, Net_Name.data());
    SERV_TCP_PORT = port_baud;

    connect_robot(Net_ID, MODEL_SCOUT, Net_Name.data(), Net_Port);

    current_vel = 0;
    current_accel = 100;

    max_vel = 100;
    max_accel = 100;

    delay.tv_sec = 1;
```

```
    delay.tv_nsec = 0;

    left_wheel = 0;
    right_wheel = 0;

    pthread_mutex_init (&Control_Lock, NULL);

    SetTimeout(2);

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

//  pthread_create( &Control_thread, &attr, Control_Loop, (void *) &max_vel);

}


Robot_Module::~Robot_Module()    {

    pthread_mutex_lock(&Control_Lock);
    pthread_cancel(Control_thread);
    pthread_mutex_destroy(&Control_Lock);

    disconnect_robot(Net_ID);

}


float Robot_Module::GetVel()     {

    return current_vel;
}


int Robot_Module::SetVel(float new_Vel)   {
    int float_max_vel;
    pthread_mutex_lock(&Control_Lock);

    if (new_Vel > max_vel) new_Vel = max_vel;
    if (new_Vel < -max_vel) new_Vel = -max_vel;

    current_vel = new_Vel;

    left_wheel = (current_vel/float_max_vel) * Robot_max_vel;
    right_wheel = (current_vel/float_max_vel) * Robot_max_vel;
    vm(left_wheel, right_wheel, 0);

    cout << endl << "New Velocity is: " << current_vel << endl;

    pthread_mutex_unlock(&Control_Lock);

    return 1;

}


int Robot_Module::Stop()         {
    int stop_vel = 0;
    pthread_mutex_lock(&Control_Lock);

    cout << endl << "Stopping...";
    st();

    current_vel = stop_vel;

    pthread_mutex_unlock(&Control_Lock);

    return 1;
}

int Robot_Module::SetTimeout(float new_time)        {
```

```
    float timeout_vel = 2.0;
    pthread_mutex_lock(&Control_Lock);

    if (new_time < 0) new_time = 1;
    else if (new_time > Robot_max_timeout) new_time = Robot_max_timeout;

    Robot_timeout = new_time;

    delay.tv_sec = (Robot_timeout / timeout_vel);
    delay.tv_nsec = (Robot_timeout / timeout_vel - delay.tv_sec) * 1000000000;

    conf_tm(Robot_timeout);

    pthread_mutex_unlock(&Control_Lock);

    return 1;

}


void *Robot_Module::Control_Loop(void *not_used)    {

    char talk[] = "Moving";
    float left_wheel = 0;
    float right_wheel = 0;
    float maxvel = 100.0;

    while(1)  {

       pthread_mutex_lock(&Control_Lock);

       left_wheel = (current_vel/maxvel) * Robot_max_vel;
       right_wheel = (current_vel/maxvel) * Robot_max_vel;
       vm(left_wheel, right_wheel, 0);
       tk(talk);

       pthread_mutex_unlock(&Control_Lock);
       sleep(2);

    }

}
```

7 Approval Sheet

This Document was produced, and approved by, the following members of the Jeah Control Systems Group:

Member Name	Member Signature
8 Hao Liang	
9 Abraham Castro Lleva	
10 Bikramjit Sarkar	
11 Emmanuel Jeremy Smadja	

12 Pledge Sheet

"On my honor as a student of the University of Virginia, I have neither given, nor received any unauthorized aid on this assignment."

Member Name	Member Signature
13 Hao Liang	
14 Abraham Castro Lleva	
15 Bikramjit Sarkar	
16 Emmanuel Jeremy Smadja	