“Let There Be Light”
EE290-O

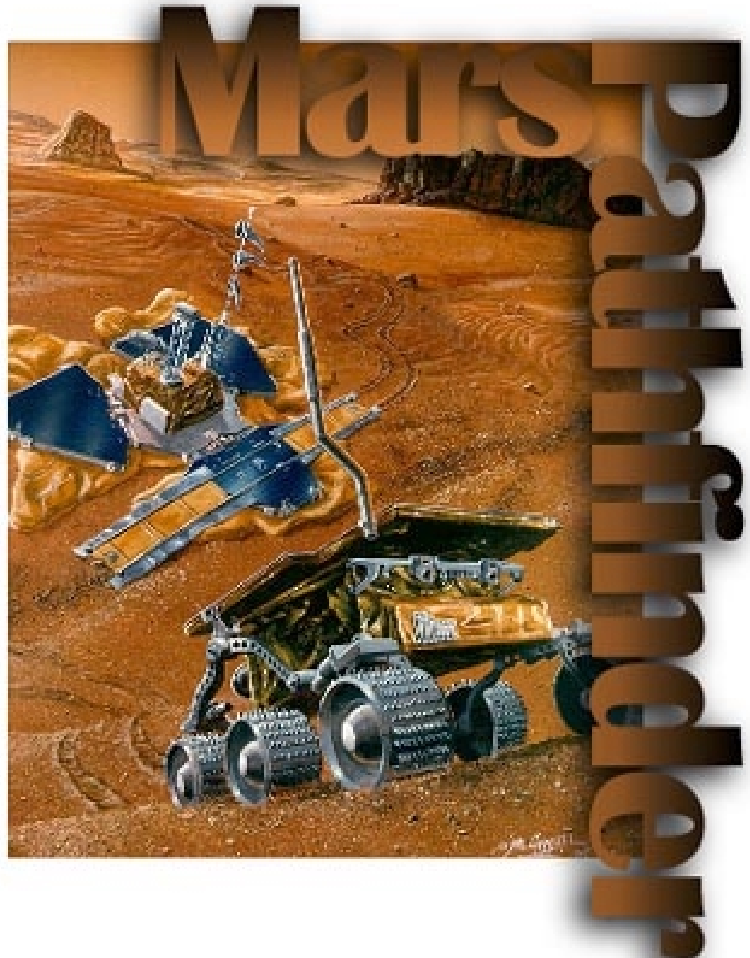
Sinem Coleri
Slobodan Matic
Anshuman Sharma

- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

Ride in the future...

- Mars Exploration
- Autonomous Unit
  - Microrover
- "Ground-truthing"
- Equipped
  - 3 scientific measuring devices



Courtesy: NASA

Precision & Accuracy

- Avenue for development of real-time applications
- Distributed tasks would be even more complex
- Example:

  A platoon of robots exploring surface chemical composition and atmospheric structure

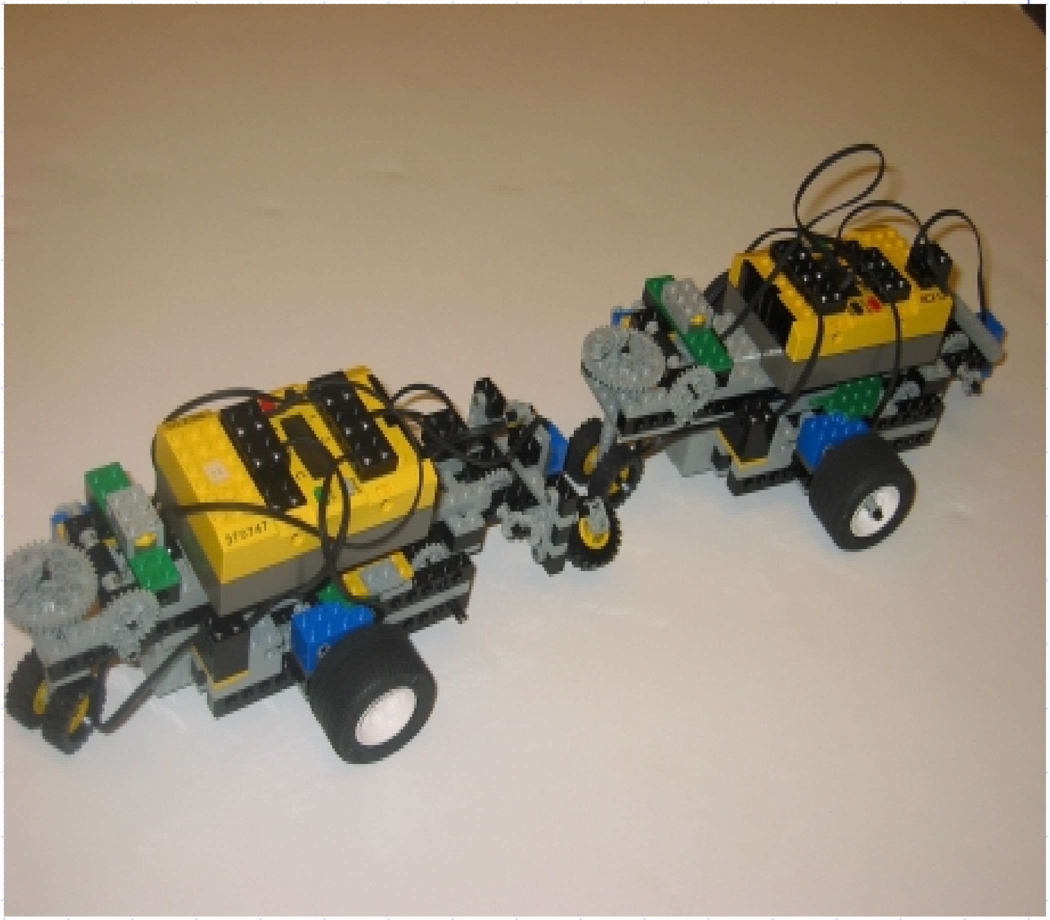
- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

Experiment

- To find an object emitting light of certain intensity
- Leader robot
  - Equipped with light sensor
- Follower
  - “Blind”, moves as directions are relayed

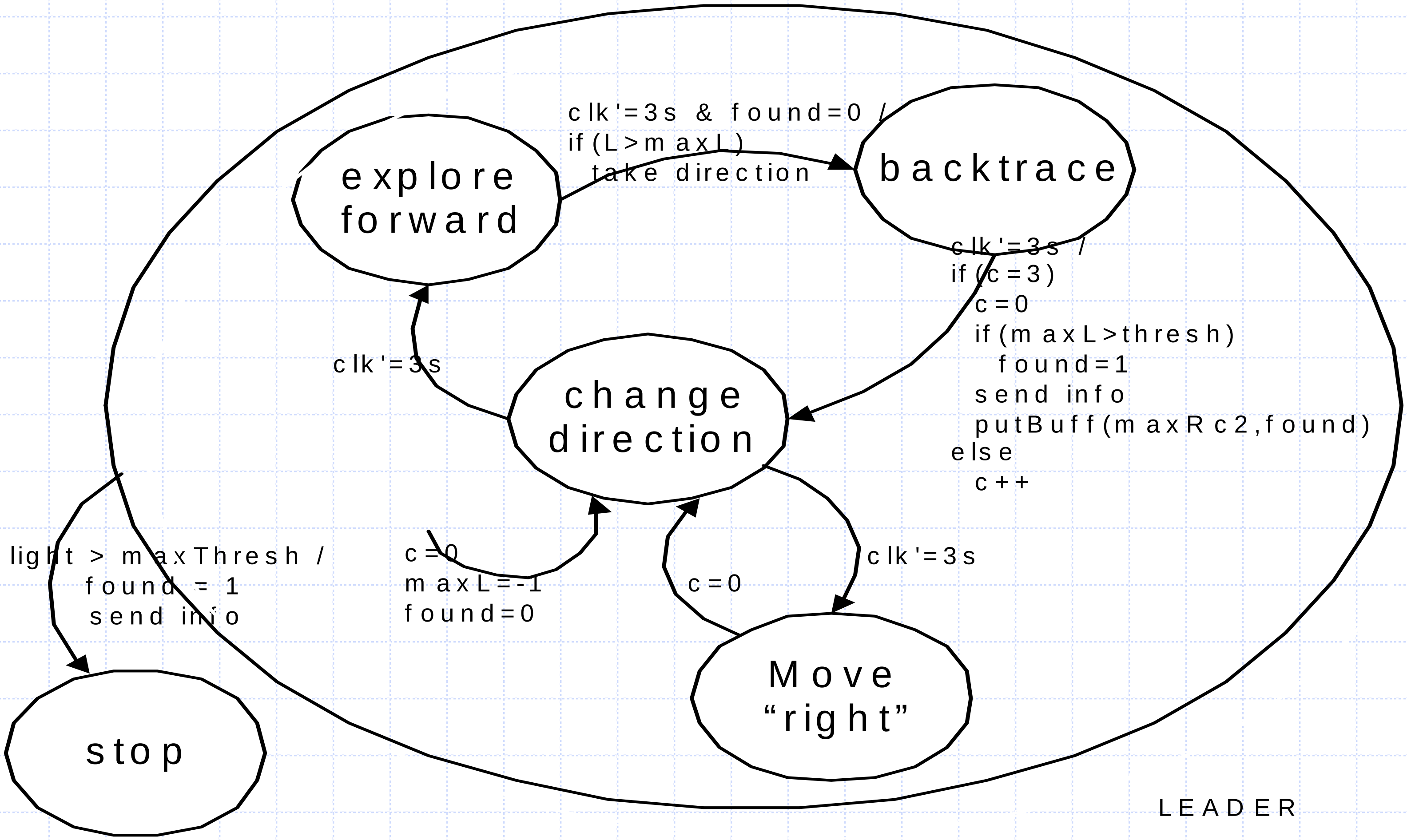
Experiment...

- Leader starts
- Identifies right direction – transmits
- Follower starts from the position just behind the leader
- Maintains a “distance” of 2 moves

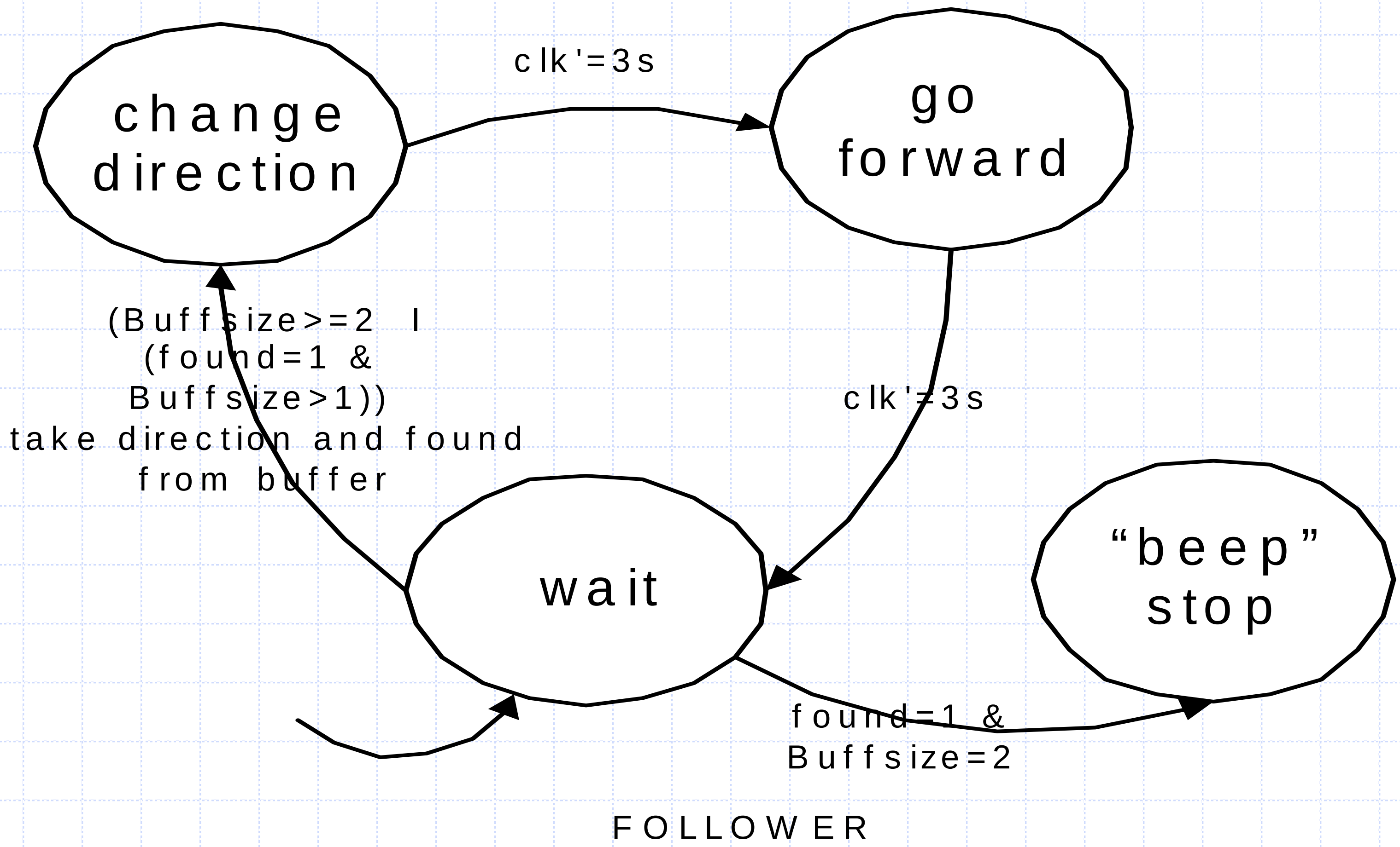


- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

Leader


explore forward
backtrace
change direction
Move “right”
stop
LEADER
clk'=3s & found=0 /
if (L>maxL)
take direction
clk'=3s /
if (c=3)
c=0
if (maxL>thresh)
found=1
send info
putBuff(maxRc2,found)
else
c++
clk'=3s
c=0
maxL=-1
found=0
c=0
clk'=3s
light > maxThresh /
found = 1
send info

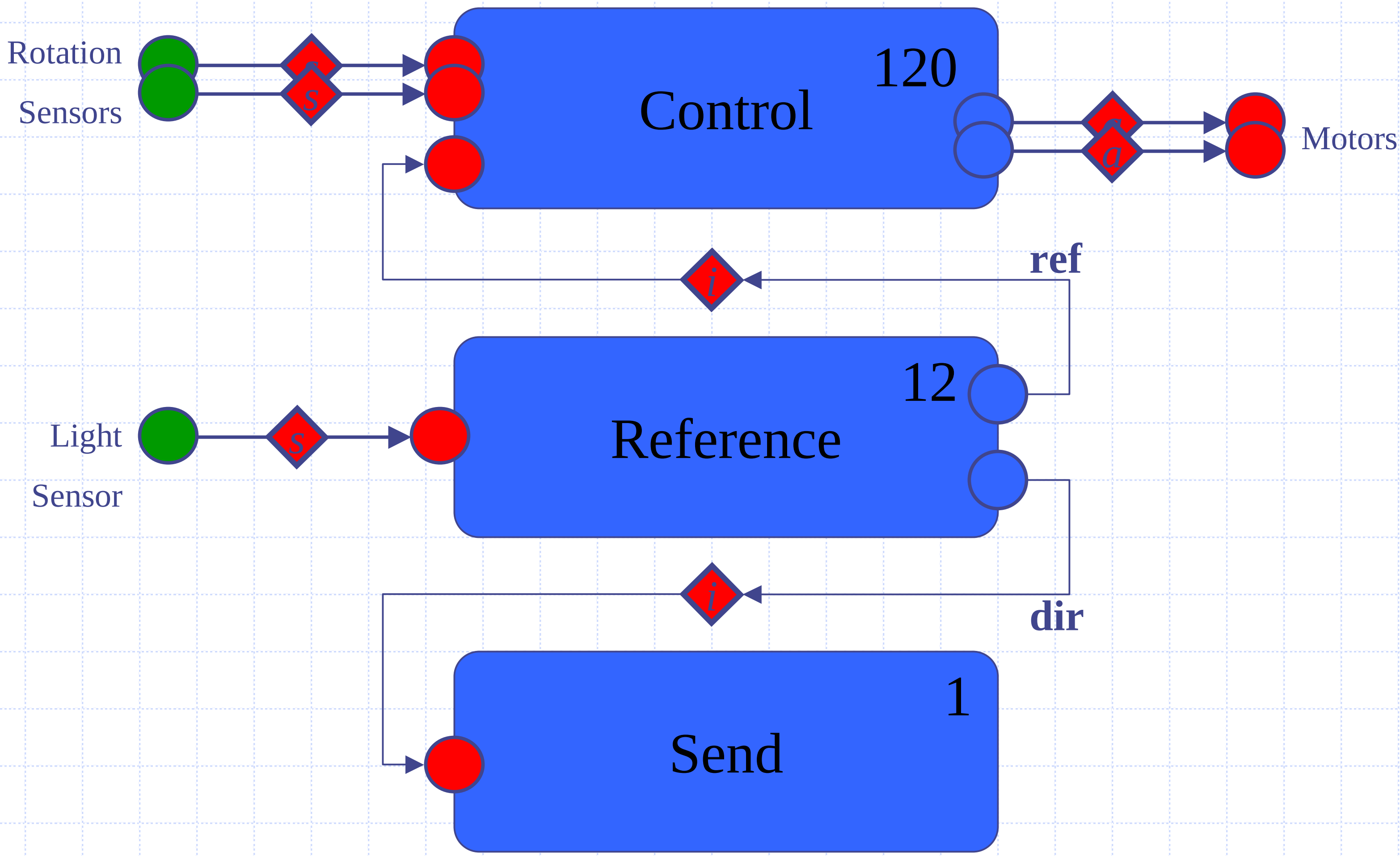
Follower


clk'=3s
change direction
go forward
(Buffsize>=2 I
(found=1 &
Buffsize>1))
take direction and found
from buffer
clk'=3s
wait
"beep"
stop
found=1 &
Buffsize=2
FOLLOWER

- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

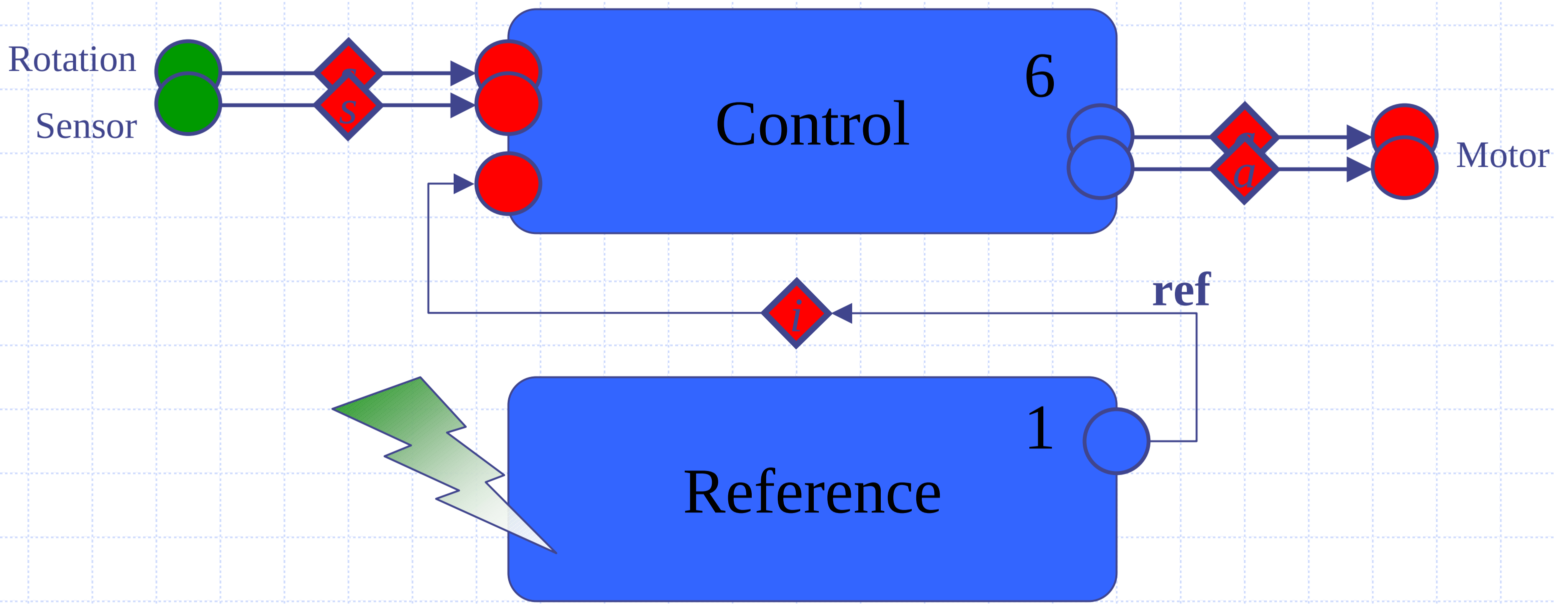
Leader program abstraction

T=36s



Follower program abstraction

T=3s



Giotto task definition

- LEADER

```
mode search() period 36000 {
    actfreq 72 do motorT(updateT_dir_speed);
    actfr eq 72 do motorR(updateR_dir_speed);

    taskfreq 72 do control(input_control);
    taskfreq 12 do reference(input_reference);
    taskfreq 1 do send(input_send);
}
```

- FOLLOWER

```
mode search() period 3000 {
    actfreq 6 do motorT(updateT_dir_speed);
    actfreq 6 do motorR(updateR_dir_speed);

    taskfreq 6 do control(input_control);
    taskfreq 1 do reference(input_reference);
}
```

Emachine

```
while (i < n_enabled_triggers) {
    /* check enabled triggers for activation */
    ...
    while (!end) { ...
        switch(e_code[pc].opcode) {
        case OPCODE_IF:
                /* evaluate condition and jump accordingly */
        case OPCODE_FUTURE:
                /* enable, insert and set trigger_time */
        case OPCODE_CALL:
                /* execute driver_code */
        case OPCODE_SCHEDULE:
                /* post task-specific semaphore */
        case OPCODE_RETURN:
                /* end == 1 */
        }
    }
```

Leader Ecode and functionality

```
instruction_t   leader[MAXINSTR] = {
        ...
  /* 11 */  CALL(T_motor_device_drv),
  /* 12 */  CALL(R_motor_device_drv),
  /* 13 */  IF(reference_cond,19),
  /* 14 */  IF(send_cond,16),
  /* 15 */  SCHEDULE(send_task),
  /* 16 */  CALL(reference_out_drv),
  /* 17 */  CALL(reference_drv),
  /* 18 */  SCHEDULE(reference_task),
  /* 19 */  CALL(control_drv),
  /* 20 */  IF(found_cond,24),
  /* 21 */  SCHEDULE(control_task),
  /* 22 */  FUTURE(500,11),
  /* 23 */  RETURN(),
  /* 24 */
        ... / * STOP mode Ecode */
}
```

```
void control_task() {
  T_speed = P*(T_g_ref - T_increment);
  T_dir = (T_speed < 0) ? fwd : rev;
  T_speed = (T_speed < 0) ? -T_speed : T_speed;

  /* the same for steering wheel */
}

void control_drv() {
  T_increment = ROTATION_3;
  R_increment = ROTATION_1;
}

void T_motor_device_drv() {
  motor_b_dir(T_dir);
  motor_b_speed(T_speed);
}

void send_task() {
    ...
  direction = read_circBuffer(&directions);
    ...
  data[0]=direction;
  data[1]=found;
    ...
  lnp_addressing_write(data,len,DEST_ADDR,MY_PORT;
}
```

Follower Ecode and functionality

```
instruction_t    follower[MAXINSTR] = {
  /* 0 */   CALL(T_motor_device_drv),
  /* 1 */   CALL(R_motor_device_drv),
  /* 2 */   IF(reference_cond,8),
  /* 3 */   CALL(reference_out_drv),
  /* 4 */   IF(found_cond,12),
  /* 5 */   IF(move_cond,8),
  /* 6 */   CALL(reference_drv),
  /* 7 */   SCHEDULE(reference_task),
  /* 8 */   CALL(control_drv),
  /* 9 */   SCHEDULE(schedule_task),
  /* 10 */  FUTURE(500,0),
  /* 11 */  RETURN(),
          ... / * STOP mode Ecode */
};
```

```
unsigned move_cond() {
    if ((directions.tail - directions.head) %
       SIZE_CIRCULAR) >= INTERVAL)  {
       return 0;
   return 1;
}

void reference_drv() {
   dirNew = read_circBuffer(&directions);
}

void reference_task() {
  switch(++search) {
   case 1:
     if(dirNew > dir)  R_I_ref = -R;
     if(dirNew < dir)  R_I_ref = R;
     break;
   case 2:
     if((dirNew-dir) > 1)  R_I_ref = -R;
     if((dirNew-dir) < -1)  R_I_ref = R;
     break;
   case 3:
     T_I_ref = T;
     dir = dirNew;
     search= 0;
     break;
   }
 }
```

- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

Communication in LegOS

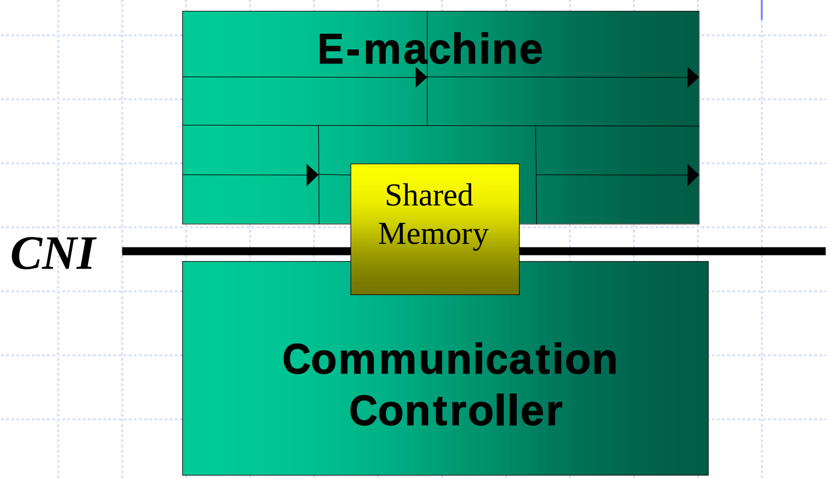
- Sending a packet
  - Send function
    - result = lnp_addressing_write(data, len, DEST_ADDR, MY_PORT)

Communication in LegOS

- Receiving a packet
  - Registering handler function with LNP:
    - lnp_addressing_set_handler(MY_PORT, packet_handler)
  - Defining handler function called from LegOS interrupt routines
    - void packet_handler(data, length, src) {……}

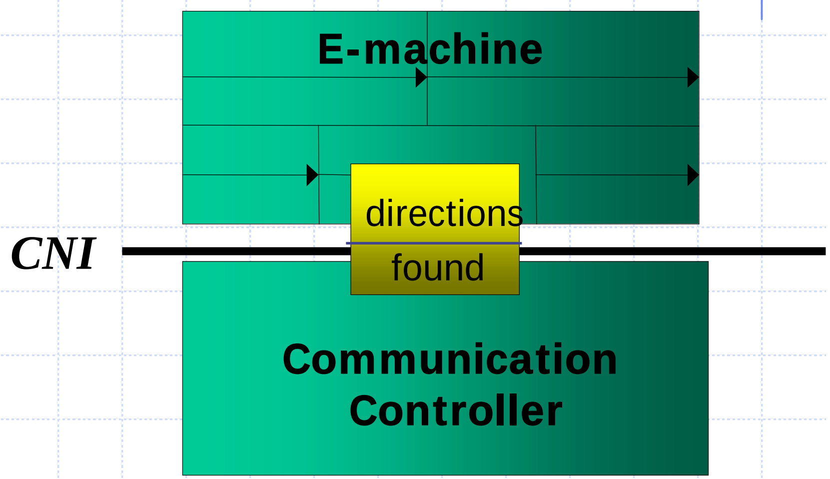
Communication-Network Interface(CNI)

- Data-sharing interface between e-machine and communication
  - Originally interface between host and TTP
    - Communication related status info in shared memory
      - Ex:global time
    - Non-blocking Write (NBW) protocol for consistent data transfer from network to host



Communication-Network Interface(CNI)

- Our interface between e-machine and communication controller
  - Movement and light info in shared memory
    - Variable *found* showing whether light is found
    - Variable keeping *directions* that leader robot has gone through

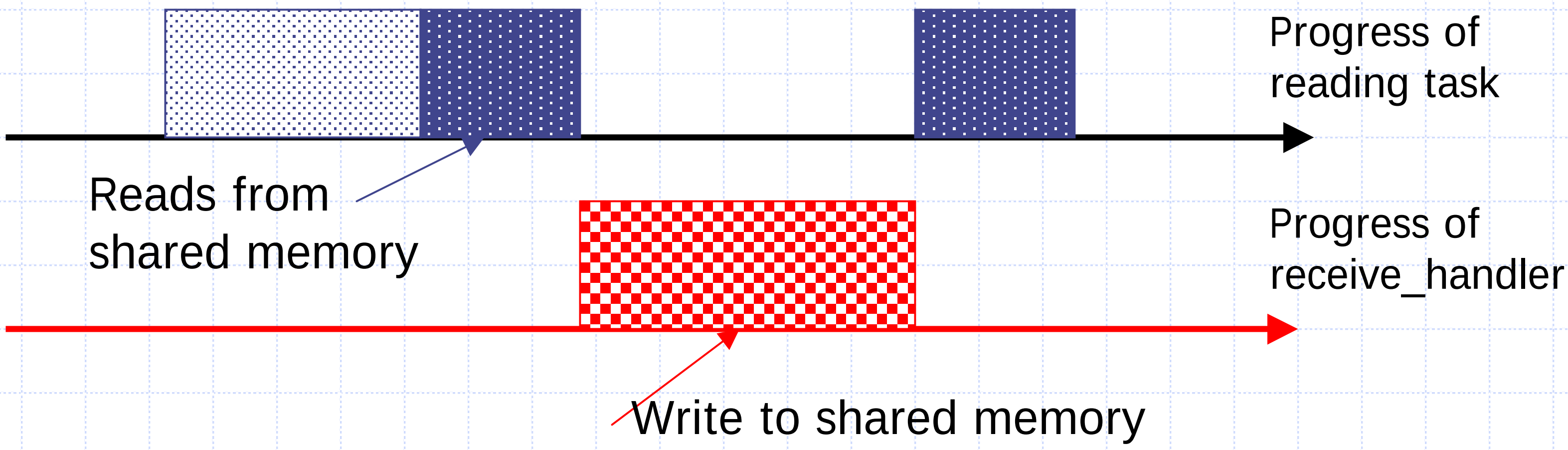


Non-Blocking Write(NBW) Protocol

- Motivation
  - Variable *found*
    - Should not be accessed from communication controller and e-machine at the same time
    - Should be updated in receiver handler without delay
      - Handler is called from interrupt routine

Non-Blocking Write(NBW) Protocol

- Writer is never blocked
  - Writes new version of data into shared memory whenever a new message arrives
- Reader retries read operation upon detection of write



Non-Blocking Write(NBW) Protocol

- Implementation
  - Concurrency control field(CCF)
    - Associated with each data type

```
typedef struct nBlockingType{
char value;
int CCF;
}nonBlockingType;
```

Non-Blocking Write(NBW) Protocol

- Implementation
  - Concurrency control field(CCF)
    - Used to detect the data access of writer

```
void write(nonBlockingType *
var, char val) {
……..
old_CCF = var->CCF;
var->CCF = old_CCF + 1;
DATA ACCESS
 var->CCF = old_CCF + 2; }
```

```
char read(nonBlockingType * var) {
…
 do {
 CCF_begin = (var->CCF);
 while ( (CCF_begin%2) == 1)
    CCF_begin = (var->CCF);
DATA ACCESS
 CCF_end = (var->CCF);
} while(CCF_end != CCF_begin);
…
```

Circular Buffer

- Motivation
  - Follower robot waits until a specific number of directions are accumulated
    - To keep the distance between leader and follower constant
  - Leader robot can get more than one correct direction before performing send operation

Circular Buffer

- Implementation
  - Classical circular buffer description →

```
typedef struct cBuffer{
 char buffer[SIZE];
 int head;
 int tail;
 }circularBuffer;
```

Circular Buffer

- Implementation
  - Reader never reads the same memory location as writer writes into

```
void write_circBuffer(circularBuffer * c, char val){
…….
if (c->head != ((c->tail + 1) % SIZE)){
c->buffer[c->tail] = val;
c->tail = (c->tail + 1) % SIZE;}
…..
```

```
char read_circBuffer(circularBuffer * c){
…..
if(c->head != c->tail){
ret = c->buffer[c->head];
c->head = (c->head + 1) % SIZE;
……
```

- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

Not a “cake-walk”

- Dependence on “quality” of batteries
- Rotational errors – poor gear mechanics
- Light source and light sensor
- Communication requires line of sight

Solutions

- Change batteries periodically
  - Tweak code to reflect battery strength
- Manually adjust gears to turn equally
  - Leader uses more battery power
- Different light orientations
  - Active vs Passive sensing
- Limit the distance between the bots
  - Communicate without line of sight!!!

- Motivation & Problem Definition
- Setup
- Algorithm Overview
- Details
- Communication
- Problems
- Conclusion

Phew!

- Spent more time calibrating than developing logic
- Algorithm modifications
  - Leader smarts
- LegOS needs power management!!