Daniel Schäfer (2549458)
Embedded Systems Milestone 3 Rafael Dewes (2548365)
Hardware Number 1 Kevin Müller (2550062)

General Scheduling Approach

The underlying task of both robots in our model is going to be the task "Orientation and Drive". It is responsible for the robots' movement and will be running whenever there is no other task in the queue. Every $\mathcal{X}$ time units this queue is checked. If there is a more urgent task present, we preempt into this new task. We assume most of our tasks are very light with respect to computation costs, thus this task will be run most of the time.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

Scout Tasks

The Scout robot has 7 distinct tasks as seen below. We chose a strict ordering of priorities depending on the urgency of each task. This will make it easier for the scheduler to choose the next task to be performed. Most tasks are not preemptive because they have very short runtime and a preemption would cause unnecessary overhead. The following table shows urgency in descending order and cost of each task, cost being a rough estimation of required operations.

	Urgency	Cost
Task 7: Receive Referee Communication	7	7
Task 3: Communicate Harvest Coordinates	6	5
Task 2: Handle High Readings	5	4
Task 1: Receive Photosensor Readings	4	10
Task 6: Communicate Own Position	3	5
Task 5: Generate New Goal	2	12
Task 4: Orientation and Drive	1	inf

Task 1: Receive Photosensor Readings We receive the readings of the photosensors via the ADC, and check them against a threshold. If treshold is met, we save current brightness value in memory; queue handling task.

- **periodic**, as we need to check if the robot is in the light in regular intervals
- **non-preemptive**, dependent on current state of the robot, important for main functionality, initiates follow up task, so its completion should not be postponed
- **not known beforehand**, we can't foresee the light behaviour, therefore the cost of this task can change depending on the brightness value perceived by the sensor. The release time and period will stay the same either way.
- **period and release time** can be the same, we assume about [10ms] but need experience with the actual hardware to fine-tune
- **approximate cost**: we get 4 values for each of the sensors (4) and need to compare each to the threshold (4), possibly write one value into memory and queue additional handling task (2). 10
- **urgency**, very urgent, should only be delayed if a dependent task is currently queued or if communication is being received
- **dependencies**: not dependent on other tasks

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

Task 2: Handle High Readings This task is generated if the value the photosensors output meets the determined threshold. It takes the current position of the robot and writes it into memory, then queues the communication.

- **aperiodic**, only triggered by preceding task
- **non-preemptive**, the current position is time critical and should not be postponed to get the most accurate and direct data, essential for the functionality of the Scout
- **not known beforehand**, as it is queued by other task we can't know exact when this task will occur
- **release time**: since the earliest we can get a new position that we want to send is after [10ms], our release time is at least that
- **approximate cost**: get current coordinates (2) and write them into a secure memory (2). 4
- **urgency**, very urgent, should again only be delayed if dependent task is
- **dependencies**: dependent on the "Receive Photosensor Readings" task for adding it to the schedule; also dependent on the basic driving task "Orientation and Driving" for position data

Task 3: Communicate Harvest Coordinates This task is generated after the photosensor values meet the threshold and the current position is saved. It takes the saved position in the memory and communicates it to the Collector via the RF module.

- **aperiodic**, only triggered by preceding task
- **non-preemptive**, the communication of this position is important and should not be postponed, essential for the functionality of the Scout
- **not known beforehand**, as it is queued by other task
- **release time**: since the earliest we can get a new position that we want to send is after [10ms], our release time is at least that
- **approximate cost**: read coordinates from memory (2) and instruct RF module with communication (3). 5
- **urgency**, very urgent, should only be postponed to correct internal positions
- **dependencies**: dependent on the "Receive Photosensor Readings" and "Handle High Readings" tasks for adding it to the schedule and saving the position coordinates

Task 4: Orientation (Theta Correction) and Drive The main task concerning movement of the robot. We adjust the orientation of our robot by changing the current

velocities of the two motors, to point towards current destination. If no adjustment is required we drive straight. When "out of bounds" is received, both motors are turned to zero for the required time. This tasks queries the "generate new destination" task upon reaching the current destination or if there is none.

- **aperiodic**, only one instance of this task is ever created
- **preemptive**, very basic task with lengthy execution. Must be preemptive for the robot to do anything but drive around.
- **not known beforehand**, arguably not entirely known as destinations are generated by other task
- **release time**:task is immediately available
- **approximate cost**: this task is greatly dependent on the physical properties of the robot, and as such is very slow and very costly time wise. We assign infinite time for this task as it will be running indefinitely, being interrupted by other tasks as needed.
- **urgency**, lowest urgency. The only risk of this task being delayed would be leaving the arena after missing the destination, but that would mean an extreme delay which seems unlikely considering the other tasks.
- **dependencies**: dependent on the "Generate New Goal" task to define the destination position.

Task 5: Generate New Goal Generate new destination where to drive, depending on the implemented strategy. At the moment it's random coordinates being generated. The destination could also follow a pattern or be somehow randomized when idly driving around.

- **aperiodic:** This task is executed once in the beginning and then gets triggered by Task 4.
- **preemptive:** Since this task depends on the tactic we implement, it could be arbitrarily long. At the same time, it is not critical for time constraints and can thus be preemted to execute other more urgent tasks.
- **not known beforehand:** We neither know when the task will be triggered nor how long it will run before outputting a new destination.
- **release time:** Becomes available as soon as enqueued by other task.
- **approximate cost:** The cost highly depends on the tactic implemented. For now, we will just randomly generate valid destinations. A simple implementation using the clock (1) and some arithmetic operations (5) could generate such destinations for each x and y. 12

- **urgency:** It is never a problem when the task gets delayed for some time because the robot will never drive blindly without having a destination. So, when no new destination gets generated for some time the robot will simply remain at its position. The only loss would be that we can not execute our tactic but that is not a big problem compared to hitting team mates or crossing the arena border. Therefore, just as Task 4 this task gets an urgency of 2.
- **Dependencies:** This task is executed when the current destination is reached and enqueued by Task 4 "Orientation and Drive".

Task 6: Communicate Own Position The Scout communicates its own position to the Collector in regular intervals as to not get pushed by its teammate.

- **periodic**, because we need to let our teammate know where we are regularly
- **non-preemptive**, dependent on current state of the robot, important for communication
- **known beforehand**, since this task will be performed in regular intervals with the only change being the robot's current position.
- **period and release time** can be the same, as we want to communicate in a regular interval. We assume about [0.5s] is enough but need experience with the actual hardware to fine-tune
- **approximate cost**: we get the current position coordinates (2) and initiate communication of these via the RF module (3), low cost of 5.
- **urgency**: this task is relatively urgent to prevent our Collector from pushing the Scout. But since it will run in regular intervals, a small delay won't be too grave.
- **dependencies**: It is dependent on the current position, so "Orientation and Driving".

Task 7: Receive Referee Communication The robot can receive its correct position in the arena from the referee. If a message was received the RF module sends an interrupt to queue this task. The Scout handles this information by updating the internal records and continuing with the updated coordinates.

- **aperiodic**, triggered by interrupt from the RF module
- **non-preemptive**, to be completed as soon as possible, s.t. the internal coordinates are accurate. Also very short task, so little delay for other tasks.
- **not known beforehand**, triggered by external source
- **release time**: Immediately available after interrupt

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

- **approximate cost**: Receiving the correct coordinate data (x,y,angle; 3) from RF module (1) and updating them internally (3) has a low cost of 7.
- **urgency**:extremely urgent, we want to minimise the time we are using inaccurate coordinates for our position calculation. Since it is triggered by hardware interrupt we can make it the most urgent task.
- **dependencies**: no dependencies

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

Collector Tasks

We divided the job of the collector into six tasks that need to be executed at different times during the game. We assigned each of them a distinct urgency which is a value between 1 and 6, 6 being the task with the highest urgency. Costs are approximated as instructions needed to do the calculations. Those might be very rough estimated but the ordering of the tasks given by the cost values should be correct and proportional to the cost of the final implementations of the tasks. The task list, ordererd by urgency:

	Urgency	Cost
Task 5: Receive Referee Communication	6	5
Task 6: Receive Harvesting Position	5	5
Task 2: Handle High Readings	4	10
Task 1: Receive Proximity Sensor Readings	3	8
Task 4: Generate New Goal	2	12
Task 3: Orientation and Drive	1	inf

Task 1: Receive Proximity Sensor Readings Reads the readings of the sensors via the ADC and compares them to the defined threshold. If a value is above the threshold, we save it in memory and enqueue the handling task (Task 2).

- **Periodic:** We have to read the sensors on a regular basis. Since this tasks is not spawned by some other task, we can execute it in regular intervals with a fixed period.
- **Non-Preepmtive:** The whole execution, including the queueing of the next task depends on the state of the robot at the exact moment where the sensor readings surpass the threshold. It is therefore important to finish the task in one go to send very up-to-date data to the follow-up task.
- **Not Known Before-hand:** The tasks takes longer when the sensor readings surpass the threshold for which we do not know when it happens statically before-hand.
- **Release time:** Same as the period (immediately available).
- **Period:** Since this task can be handled very fast, the period can also be very fast. We will start with a period of 10ms and later fine-tune it using the hardware.
- **Relative deadline:** Does not need a deadline
- **Approximate cost:** The following operations need to be executed where the last one needs to be done only when the readings are higher than the threshold, which should happen rather rarely: Read each sensor (3) and for each one compare the value to the threshold (3), enqueue task (i.e. make call to the scheduler) to handle high readings (2). Therefore we assign this task a (relative) cost of 3+3+2=8.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

- **Urgency:** The worst thing that happens when the task is delayed for too long it that he robot might not see another robot nearby. This does not do any damage. However, it might also collide with its team mate which would do significant damage so this task has a (relative) urgency of 3.
- **Dependencies:** No dependencies on other tasks.

Task 2: Handle High Readings Receives the sensor direction and value of the last reading that was above the threshold (not that this task is spawned by Task 1). It tries to estimate the position of the team mate and decide whether the sensor reading could be created by the team mate. If yes, it continues with the current destination, otherwise it will try to hit the sensed target by updating the destination value.

- **Aperiodic:** This task is only triggered after an above-threshold proximity sensor reading is sensed in Task 1.
- **Non-Preepmtive:** The current sensor readings depend on the current position of the collector and the nearby robots. Therefore, if the task is delayed, the values will quickly become useless. Furthermore, from our simulation we already noticed that it is important to react to proximity readings very fast in order to have a chance to hit the opponent and avoid hitting our team mate. Therefore, this task should be executed in one go.
- **Not Known Before-hand:** We can not know when the task is triggered in the previous task and also the runtime will depend on the system state.
- **Release time:** Becomes available as soon as enqueued by other task.
- **Relative deadline:** Depending on the threshold, this task should get a deadline that ensures that the collector will never push its team mate. Since we have not fixed the threshold proximity and maximum driving speed, the exact deadline can only be calculated later when working with the real hardware.
- **Approximate cost:** The following operations need to be executed: Retrieve last scout position (2), retrieve time and calculate elapsed time since last scout position update (3), calculate the distance that he scout has traveled since (i.e. the radius where scout might be) (2), compare current proximity value with possible scout position (3) and make decision whether to hit or not and depending on the result set the current destination value (1). Therefore we assign this task a (relative) cost of 1+3+2+3+1=10.
- **Urgency:** This task has about he same reasoning for urgency as Task 1. However, since this task will be triggered rather rarely, we want to make sure to act on the new data very quickly so it also gets urgency of 4.
- **Dependencies:** This task can only be executed after Task 1. This is ensured because this task is scheduled at the end of Task 1 when all read-write operations

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

to the shared global memory are done and the task is finished.

Task 3: Orientation and Drive The tasks adjusts the engine velocities in order for the robot to point towards the current destination. Upon reading the destination, a new task to generate a new destination is created.

- **Aperiodic:** There is only one permanent instance of this task running at any time.
- **Preepmtive:** This should be the default task which is executed when no other task needs to be executed. This task will be preemted whenever another more important task needs to be executed. We assume here that even with all other period tasks running, there will still be enough time left for the execution of this task.
- **Not Known Before-hand:** Execution depends on the generated destinations which are not known before-hand.
- **Release time:** Same as the period (immediately available).
- **Relative deadline:** Does not need a deadline.
- **Approximate cost:** The cost is proportional to the radius that the robot needs to turn and depends on the wheel velocities we choose. In any case, this task takes a lot longer than all other tasks because a physical motion and calculation of the new rotation needs to be done. We could set the cost of this task to be 20 per degree the robot needs to turn.
- **Urgency:** If the task gets delayed for too long we might miss the target destination and in the worst case cross the arena border. However, for this to happen the task would have to be delayed for a very long time (which is unlikely given the other tasks) which is why it will get a relatively low urgency of 1.
- **Dependencies:** The robot needs a destination before executing this task. Therefore, Task 4 has to finish before this task starts for the first time. This task queues Task 4 upon reaching the current destination.

Task 4: Generate New Goal Generate new destination where to drive (which is subject to tactic finetuning). Initially, we will use random coordinates within the arena instead of an elaborate tactic. The destination could also follow a pattern or be somehow randomized when idly driving around.

- **Aperiodic:** This task is executed once in the beginning and then gets triggered by Task 3.
- **Preepmtive:** Since this task depends on the tactic we implement, it could be arbitrarily long. At the same time, it is not critical for time constraints and can thus be preemted to execute other more urgent tasks.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

- **Not Known Before-hand:** We neither know when the task will be triggered nor how long it will run before outputting a new destination.
- **Release time:** Becomes available as soon as enqueued by other task.
- **Relative deadline:** Does not need a deadline.
- **Approximate cost:** The cost highly depends on the tactic implemented. For now, we will just randomly generate valid destinations. A simple implementation using the clock (1) and some arithmetic operations (5) could generate such destinations with a cost of 2 x (1 + 5) = 12.
- **Urgency:** It is never a problem when the task gets delayed for some time because the robot will never drive blindly without having a destination. So, when no new destination gets generated for some time the robot will simply remain at its position. The only loss would be that we can not execute our tactic but that is not a big problem compared to hitting team mates or crossing the arena border. Therefore, just as Task 4 this task gets an urgency of 2.
- **Dependencies:** This task is executed when the current destination is reached and enqueued by Task 3.

Task 5: Receive Referee Communication The robot can receive its correct position in the arena from the referee. If a message was received the RF module sends an interrupt to trigger this task. The Scout handles this information by updating the internal records and continuing with the updated coordinates.

- **Aperiodic:** Gets triggered by an interrupt from the RF module.
- **Non-preepmtive:** Driving with an inacurate position can be very bad (as seen in the simulation). Therefore, when we get the chance to correct our position we want to do this immediately and without delay. The accuracy of all other tasks depends on a correct position so no other task could justify an interruption of this task.
- **Not Known Before-hand:** We don't know when the task gets triggered. But we exactly know its runtime.
- **Release time:** Immediately available after interrupt.
- **Relative deadline:** Does not need a deadline.
- **Approximate cost:** The robot needs to read the data from the RF module (1) and extract both x and y positions as well as the angle (3). It then update its internal estimated position to match the real position (3). It therefore gets a cost of 7.
- **Urgency:** Driving with an incorrent estimation of the real position is very bad. Considering that collisions occur very often, it is extremly important to update the estimation as soon as possible. It therefore gets the highest urgency of 6.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

- **Dependencies:** No dependencies.

Task 6: Receive Harvesting Position The robot reads a new harvesting position from the RF module and updates its current destination accordingly.

- **Aperiodic:** Gets triggered by an interrupt from the RF module.
- **Non-preepmtive:** Since the task has a very low cost, we think preemting it would be too much overhead so we want to finish it in one go.
- **Not Known Before-hand:** We don't know when the task gets triggered. But we exactly know its runtime.
- **Release time:** Immediately available after interrupt.
- **Relative deadline:** Does not need a deadline.
- **Approximate cost:** The robot needs to read the data from the RF module (1) and extract both x and y positions of the harvest position (2). It also updates the current destination (2). Therefore, the cost is 5.
- **Urgency:** Since the light might move very fast, it is important to react to new harvest positions as soon as possible. When the collector waits for too long, the light might have moved somewhere else. It therefore gets a relatively high urgency of 5.
- **Dependencies:** No dependencies.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

Scheduler: Decisions and Optimizations

```
/* Struct to maintain references to the entire List */
struct task_list
  {
    struct task *head;
    struct task *tail;
    bool is_empty;
  };

/* element contained in taskList, doubly linked
   priority will be = task_identifier for now */
struct task
  {
    struct task *prev;
    struct task *next;
    enum task_identifiers_collector task_identifier;
    short priority;
  };
```

Currently the scheduler is implemented using a sorted doubly linked list mainly for convenience reasons and its flexibiliy. It does not strictly resemble any scheduler that was presented in the lecutre, as we expect a simple priority sorted queue to be more efficient for our specific scenario. Every task except for the 'default task' (orient into direction and drive) will have very small cost and very small runtime. This allows us to only schedule whenever a task has finished (except for the default task, which will be interrupted every $\mathcal{X}$ timeunits).

Insertion into the tasklist will always happen sorted by decreasing priority (which is very convenient for the double linked structure). This means that scheduling simply has to take the head of the list. All tasks except for default task will be removed from the tasklist, which means that the scheduling will always schedule the 'default task' if and only if nothing else is contained (as the default task has the lowest priority for both collector and scout).

Every task contains an identifier, which specifies which job it represents as an enum value (different jobs for scout and collector respectively). They are sorted in increasing priority, which makes it possible to additionally use the enum values as priority. These enum values could be replaced by C macros to compiler insert the identifiers as integer, which reduces the neccessary code-base.

We still decided to use a priority field for each task seperate to the idenfier to allow changing of priorities of the same task. This is currently not planned, but we decided to keep this field for now and set it equal to the identifier enum value.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

An optimization that would be applicable in the current task structure is to replace the entire linked list by an 8 bit integer using binary encoding. This is possible because our current scheduling plan will assure, that every task will be contained in the queue at most once. The task with the highest priority contained in the list would set the most significant bit to 1, not contained would set it to 0. The task with the second highest priority would set the second most significant bit to 1 if contained, 0 otherwise etc. The 'default task' (correct theta and drive) would not have to be contained in the list, as it will always be contained in the current model. This approach would work for up to 9 Tasks (including the default task, as we do not have to track if it is contained or not), which makes it applicable to our currently planned structure.

Replacing the linked list structure with an 8 bit array would significantly decrease the costs of all procedures on the structure, as every procedure would be posible with a single binary operation instead of keeping the list sorted and handling moving references to the listed objects.

We decided against this change for now, as our scheduling is already very low-cost and we have to test on the hardware first wether or not it is feasible to guarantee that no task will be contained in the list twice at any point and also how long all the non-default-tasks really take.

Testing and Specifics

Our C implementation currently works on `task_identifiers_collector` as explained below. The implemenation which would be used on the scout is exactly the same, one would simply replace all occurances of `task_identifiers_collector` with `task_identifiers_scout`

Compiling the scheduler using `make` will create a binary called `scheduler`, which can be run using `./scheduler`. This per default runs all tests (which can be disabled using the C Macro testing defined in `main.c`) and a short demonstration how this works in practice. One can change this demonstration to test any arbitrary scenario, by changing the insertions and schedule calls at the very bottom of the `main()` function in `main.c`. Instead of executing the tasks it uses suited values for sleeptime.

Daniel Schäfer (2549458)
Rafael Dewes (2548365)
Kevin Müller (2550062)

The task enums

Below you will find the possible values for all task identifiers of both scout and collector

```
/* lower task identifier -> lower priority
   New tasks are added here with INCREASING PRIORITY */
enum task_identifiers_collector
  {
    COLLECTOR_CORRECT_THETA = 001,
    COLLECTOR_GENERATE_NEW_GOAL = 002,
    COLLECTOR_RECEIVE_SENSOR_READINGS = 003,
    COLLECTOR_HANDLE_HIGH_SENSOR_READINGS = 004,
    COLLECTOR_RECEIVE_HARVESTING_POSITION = 005,
    COLLECTOR_HANDLE_REFEREE_UPDATE = 006,
  };

/* lower task identifier -> lower priority
   New tasks are added here with INCREASING PRIORITY */
enum task_identifiers_scout
  {
    SCOUT_ORIENTATION_AND_DRIVE = 001,
    SCOUT_GENERATE_NEW_GOAL = 002,
    SCOUT_COMMUNICATE_OWN_POSITION = 003,
    SCOUT_RECEIVE_SENSOR_READINGS = 004,
    SCOUT_HANDLE_HIGH_SENSOR_READINGS = 005,
    SCOUT_COMMUNICATE_HARVEST_COORDINATES = 006,
    SCOUT_HANDLE_REFEREE_UPDATE = 007,
  };
```