Master Lab IoT

Lukas Radovansky
Technische Universität München

20.1.2025

Contents

1 Background for grading (0.5 page)

- **General overview of your schedule including longer leave of absence (e.g., vacations) to explain missing data:**
  - I was on Christmas leave from 21.12.2024 to 2.1.2025.
- **Hardware problems:**
  - At the beginning of the semester, I encountered a problem with my ESPs that couldn't maintain a stable connection to the FRITZBOX router.
  - I spent a lot of time trying to fix the issue, but I couldn't find a solution.
  - In the end, I asked my landlord to give me the router credentials to debug the issue, which he refused.
  - Instead, he provided me with a small router that I connected to the FRITZBOX. This solved the issue, and I was able to continue with the project.
  - However, the router was not able to maintain a stable connection all the time and sometimes had to be restarted.
- **Credentials changed for the Digital Twin:**
  - There were no credentials changed for the Digital Twin app.

2 Your setup at home (1 page)

- **Scale map and sensor placement:**
  - Present the scale map with the placement of the sensor and the observed area.

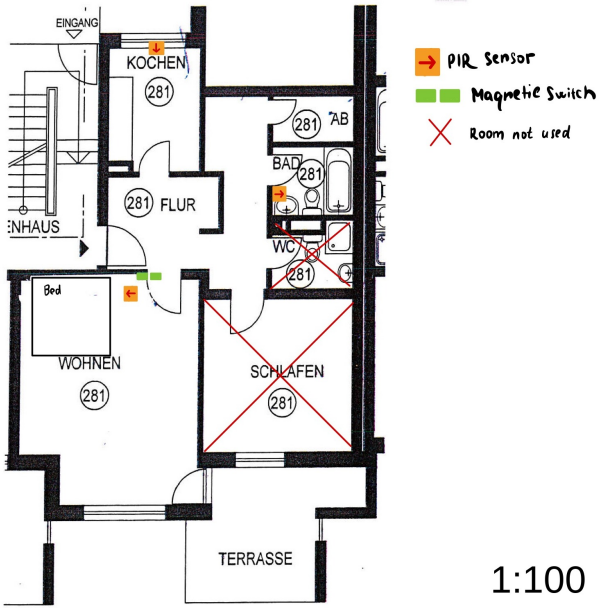

Figure 1: Scale map illustrating the sensor placement and the monitored area. The scale is 1:100. Red arrows indicate the orientation of the PIR sensors. Crossed-out areas are not part of the rented section of the flat.

- **Challenges from positioning:**
  - The Sensor in the kitchen triggered false positives if the door to the kitchen were opened and someone was in the hallway.
  - The sensor placed in bathroom could be affected by the steam from the shower, however the device seems to survive the humidity well.
  - I have deployed exactly one ESP with a Magnetic Switch and PIR sensor simultaneously in my apartment. It has been placed in the bedroom. The PIR sensor was pointing towards the bed area, and the magnetic switch was placed on the entrance door. The idea was to track the sleep time of the patient. If the doors remained open, the sensor was still active, and no bed area events were recorded. This was done on purpose because I know that every time the patient goes to sleep, he will close the door first.
- **Setup information:**
  - Sharing the flat with another person and a cat presented additional challenges in completing some tasks for this seminar.
  - Occasionally, a third person (the landlord) would be present in the flat once or twice a week to work from the crossed-out room on the apartment map. The landlord also used the kitchen, where one of the PIR sensors was placed.
  - Throughout the semester, we hosted guests on several occasions (4-5 times), which impacted the data collection process. The highest number of guests was five, during the weekend of December 12-15, 2024.

3 Sensor Development (10 pages)

3.1 Challenges (0.5p)

- **Describe any aspect to consider while grading that impacted the development of your code, such as: programming skills, difficulties in understanding concepts, etc.**
  - I had limited experience with embedded systems programming, which required additional time to learn and understand the given tasks.
  - I was also new to Docker and Kubernetes.

3.2 Sensor Integration (0.5p)

- **Integration of PIR and Magnetic Sensors**
  The integration of PIR sensors with a magnetic switch sensor was achieved through the following steps:
  1. **Hardware Configuration:**
     - **PIR Sensor** connected to GPIO pin 27.
     - **Magnetic Switch Sensor** connected to GPIO pin 33.
  2. **GPIO Initialization:** Both sensors are configured as RTC GPIOs with pull-down resistors to ensure stable low states when inactive.

```
void configure_rtc_gpio() {
    // Initialize PIR sensor GPIO
    rtc_gpio_init(27);
    rtc_gpio_set_direction(27, RTC_GPIO_MODE_INPUT_ONLY);
    rtc_gpio_pulldown_en(27);

    // Initialize Magnetic Switch GPIO
    rtc_gpio_init(33);
    rtc_gpio_set_direction(33, RTC_GPIO_MODE_INPUT_ONLY);
    rtc_gpio_pulldown_en(33);
}
```

3. **Wake-Up Configuration:** Both sensors are set as wake-up sources using the EXT1 mechanism, allowing the ESP32 to wake from deep sleep when either sensor is triggered.

```
uint64_t wakeup_pins = (1ULL << 27) | (1ULL << 33);
esp_sleep_enable_ext1_wakeup(wakeup_pins, ESP_EXT1_WAKEUP_ANY_HIGH);
```

4. **Event Handling:** Upon wake-up, the system identifies which sensor triggered the event and processes it accordingly.

```
void handle_wakeup_reason(){
    if (wakeup_reason == ESP_SLEEP_WAKEUP_EXT1) {
        uint64_t status = esp_sleep_get_ext1_wakeup_status();
        if (status & (1ULL << 27)) {
            // Handle PIR sensor event
        }
        if (status & (1ULL << 33)) {
            // Handle Magnetic switch event
        }
    }
}
```

3.3 Single Code for All Sensors (1p)

- **Generic Code Base for Multiple Sensors**
  - **Describe how you achieved a generic code base for all your sensors devices. You can include code snippets.**
    - Generic code for all devices was achieved by identifying the device based on its MAC address and configuring it accordingly.
    - Each device needs to be specied in the ‘main.h‘ file with its device name, MAC address, ID, MQTT topic, security key, battery information availability, and room ID.

```
//Struct definition for device information in main.h
/**
 * @brief Represents the configuration and metadata for a device.
 *
 * This struct is used to define the properties of an ESP device, including its
 * name, MAC address, ID, MQTT topic, security key, and whether battery information
 * is available. The struct can be used to identify devices and manage their specific
```

```
 * configurations within the system.
 */
 typedef struct {
    char* device_name;             // < Name of the device (e.g., "Living Room").
    uint8_t mac_address[6];        // < MAC address of the device (6 bytes).
    int device_id;                 // < Unique identifier for the device.
    char* device_topic;            // < MQTT topic for publishing device data.
    char* device_key;              // < Security key for authenticating with the MQTT broker.
    bool battery_info_available;   // < Indicates if the device provides battery information.
    char* room_id;                 // < Id of the room as named in the InFlux database.
} device_info_t;


// Example of definition of the device in main.h
// (name, mac adress, topic, key, battery info available, room_id)
#define ESP_DEVICE_1 {"Living Room", {0xEC, 0x62, 0x60, 0xBC, 0xE8, 0x50}, 4,
"1/4/data", "key", true, "livingroombedarea"}

// Read the MAC address and identify the device in the main function:
uint8_t mac_address[6];
esp_read_mac(mac_address, ESP_MAC_WIFI_STA);
identify_device(mac_address);

// The implementation fo the identify_device function in main.c
identify_device(mac_address);
/**
 * @brief Identifies the current device based on its MAC address.
 *
 * Matches the MAC address of the device with the pre-configured device list in `main.h`.
 * Sets the device's ID, MQTT topic, security key, and battery information availability.
 *
 * @param mac_address Pointer to the MAC address array of the device.
 */
void identify_device(const uint8_t* mac_address) {
    for (int i = 0; i < sizeof(ESPs) / sizeof(ESPs[0]); ++i) {
        if (memcmp(mac_address, ESPs[i].mac_address, sizeof(ESPs[i].mac_address)) == 0) {
            // Copy the device info into this_device
            memcpy(&this_device, &ESPs[i], sizeof(device_info_t));
            // Logging
            ESP_LOGI("*", "********** Device identified as %s", this_device.device_name);
            return;
        }
    }
    ESP_LOGI("*", "Device not recognized.");
}
```

3.4 Wake-up stub (2p)

3.4.1 Key Points

- **Sensor Trigger Filtering:** It checks if the sensors are newly triggered or still active. If the trigger is repetitive within a short time window, it returns to deep sleep without a full wake-up.

```
if (my_rtc_time_get_us() / 1000000 - last_wakeup_RTC <= SENSOR_INACTIVE_DELAY_IN_WAKE_UP_STUB_SEC) {
        last_wakeup_RTC = my_rtc_time_get_us() / 1000000;
        ets_delay_us(1000000); // Delay in microseconds.
        esp_wake_stub_set_wakeup_time(AUTOMATIC_WAKEUP_INTERVAL_SEC * 1000000);
        // Set stub entry, then go to deep sleep again.
        esp_wake_stub_sleep(&wake_stub);
}
last_wakeup_RTC = my_rtc_time_get_us() / 1000000;
```

- **Conditional Full Wake-Up:** If the PIR event array is full, or if a magnetic switch triggers, the stub decides to fully boot into the main application.
- **Scheduled Timer Wake-Up:** Before returning to deep sleep, it sets a next wake-up timer, thus allowing periodic checks without fully waking the system each time. For this the variable `AUTOMATIC_WAKEUP_INTERVAL_SEC` is used to set the next wake-up time.

3.4.2 Timestamp Handling and Sending PIR Events in Time

- **RTC-Based Timing:** A helper function converts RTC counter ticks into microseconds, which the stub uses to get a "local RTC time" even in deep sleep.

```
/**
 * @brief Retrieves the current RTC time in microseconds.
 *
 * This function reads the RTC time registers to obtain the current time.
 * It operates in the wake-up stub environment and uses low-level register access.
 *
 * @return Current RTC time in microseconds.
 */
RTC_IRAM_ATTR uint64_t my_rtc_time_get_us(void)
{
    SET_PERI_REG_MASK(RTC_CNTL_TIME_UPDATE_REG, RTC_CNTL_TIME_UPDATE);
    while (GET_PERI_REG_MASK(RTC_CNTL_TIME_UPDATE_REG, RTC_CNTL_TIME_VALID) == 0) {
        ets_delay_us(1); // Wait for RTC time to be valid.
    }
    SET_PERI_REG_MASK(RTC_CNTL_INT_CLR_REG, RTC_CNTL_TIME_VALID_INT_CLR);
    uint64_t t = READ_PERI_REG(RTC_CNTL_TIME0_REG);
    t |= ((uint64_t)READ_PERI_REG(RTC_CNTL_TIME1_REG)) << 32;

    uint32_t period = REG_READ(RTC_SLOW_CLK_CAL_REG);

    // Convert RTC clock cycles to microseconds.
    uint64_t now_us = ((t * period) >> RTC_CLK_CAL_FRACT);
```

```
    return now_us;
}
```

- **Unix Timestamp Calculation:** After each successful SNTP sync in the main app, we store:
  - `rtc_time_at_last_sync`: RTC time in milliseconds at sync.
  - `actual_time_at_last_sync`: Real Unix epoch time in milliseconds.

```
// Extern declarations for time synchronization variables during wake up stub in main.h
// These variables will be stored in RTC memory to persist across deep sleep cycles
extern RTC_DATA_ATTR uint64_t rtc_time_at_last_sync;
extern RTC_DATA_ATTR uint64_t actual_time_at_last_sync;

// Storing reference times in the main app (after SNTP sync)
// main.c - after SNTP synchronization
actual_time_at_last_sync = get_current_time_in_ms();
rtc_time_at_last_sync = get_time_since_boot_in_ms();

uint64_t get_current_time_in_ms() {
    struct timeval now;
    gettimeofday(&now, NULL);
    return (uint64_t)(now.tv_sec) * 1000 + (now.tv_usec) / 1000; // Convert to milliseconds
}

uint64_t get_time_since_boot_in_ms() {
    return (my_rtc_time_get_us() / 1000); // Convert microseconds to milliseconds
}
```

The wake stub computes the *actual* timestamp by adding the difference between current RTC time and `rtc_time_at_last_sync` to `actual_time_at_last_sync`.

```
// Computing actual timestamp in the wake stub:
void store_pir_event(void)
{
    // 1. Current RTC time in ms
    uint64_t rtc_time_now = my_rtc_time_get_us() / 1000;

    // 2. Calculate the time difference since last sync
    uint64_t rtc_time_diff = rtc_time_now - rtc_time_at_last_sync;

    // 3. Compute actual Unix timestamp in ms
    uint64_t actual_timestamp = actual_time_at_last_sync + rtc_time_diff;

    // Store the PIR event with the computed timestamp
    pir_events[pir_event_count].timestamp = actual_timestamp;
    // ...
}
```

- **PIR Event Storage:** The PIR events are stored in an array of structs in RTC memory, until the array is full of a magnetic switch triggers a full wake-up or automatic wake-up interval is reached.

```c
/**
 * @brief Represents the struct for a PIR event
 *
 * This struct includes the timestamp of the event and the device information.
 */
 typedef struct {
    uint64_t timestamp;       // The actual Unix timestamp in milliseconds
    device_info_t device;     // Device information associated with the event
} PIR_Event_t;

// Define the maximum number of PIR events stored in RTC memory, the array
// to store these events, and the counter for the events in main.h
extern RTC_DATA_ATTR uint32_t MAX_PIR_EVENTS;
extern RTC_DATA_ATTR PIR_Event_t pir_events[CONFIG_MAX_PIR_EVENTS];
extern RTC_DATA_ATTR int pir_event_count;
```

– If a PIR sensor wakes the device, the stub calls:

```c
/**
 * @brief Stores a PIR event with the current timestamp and device information.
 *
 * Calculates the actual timestamp based on RTC time and synchronization data,
 * then stores the event in the PIR events array.
 */
 void store_pir_event(void)
{
        //...

    // Store the new event with the calculated actual timestamp.
    pir_events[pir_event_count].timestamp = actual_timestamp;

    // Since we cannot use memcpy in the wake-up stub, we manually copy each field.
    pir_events[pir_event_count].device.device_id = this_device.device_id;
    pir_events[pir_event_count].device.battery_info_available = this_device.battery_info_available;

    // Note: Assigning pointers directly as below is acceptable in the wake-up stub environment.
    pir_events[pir_event_count].device.device_name = this_device.device_name;
    pir_events[pir_event_count].device.device_topic = this_device.device_topic;
    pir_events[pir_event_count].device.device_key = this_device.device_key;
    pir_events[pir_event_count].device.room_id = this_device.room_id;

    // Copy MAC address manually.
    for (int i = 0; i < 6; i++) {
        pir_events[pir_event_count].device.mac_address[i] = this_device.mac_address[i];
    }

    // Increment the event count.
    pir_event_count++;
}
```

– If `pir_event_count` hits its maximum, the stub forces a full wake to upload all stored events via MQTT.

3.5 Sending Battery RSOC (0.3p)

- **RTC Timestamp Check:** The stub keeps a variable `last_battery_info_time_RTC`. On each wake, it checks if enough time has passed:

```
if ((my_rtc_time_get_us() / 1000000) - last_battery_info_time_RTC
        >= BATTERY_INFO_INTERVAL_SEC) {
    last_battery_info_time_RTC = my_rtc_time_get_us() / 1000000;
    esp_default_wake_deep_sleep(); // Force main app to fully boot
}
```

- Once awake, the main application reads the battery gauge and sends the RSOC (Remaining State of Charge) via MQTT. Since `last_battery_info_time_RTC` is in RTC memory, it persists through deep sleeps, ensuring strictly periodic RSOC uploads.

3.6 Monday Problem (0.3p)

- **What is the Monday problem?**
  The "Monday problem" refers to a situation where the Raspberry Pi (running the MQTT broker) is physically taken to class on Mondays, making the broker temporarily unreachable.
- **Approach to solving the Monday problem**
  To handle the broker's unavailability, I introduced a boolean variable, `mqtt_broker_connected`, in `mqtt.c`. Whenever the device cannot connect to the broker, it skips sending data and retains the events in RTC memory. The next time the device fully wakes and finds the broker reachable, it sends all stored data.
- **Alternative attempts?**
  I did not attempt alternate solutions because the above approach—caching unsent events until reconnection—proved both straightforward and reliable.

```
// Wait for connection with a timeout of 10 seconds
EventBits_t bits = xEventGroupWaitBits(
    mqtt_event_group, CONNECTED_BIT, false, true, pdMS_TO_TICKS(10000)
);

if (bits & CONNECTED_BIT) {
    ESP_LOGI("mqtt", "Connected to MQTT\n");
} else {
    ESP_LOGI("mqtt", "Could not connect to MQTT broker\n");
    mqtt_broker_connected = false;
}

// Example of error handling for PIR events
//(for battery information and magnetic switch data is the approach same)
if (!mqtt_broker_connected) {
    ESP_LOGI("PIR", "Cannot send stored PIR events, MQTT is not connected");
    return;
}
```

3.7 Power Consumption (1p)

- Provide the updated formula to estimate the battery life and the necessary measured data.

3.8 Limitations (0.5p)

The final solution meets the core requirements. However, it can be improved in the following ways:

- Currently, only simple logic is used to filter repeated sensor triggers (If the patient stays active next to the sensor). More sophisticated signal filtering could prevent repeated events.
- The application relies heavily on RTC memory for event storage. If the memory becomes full, older events risk being overwritten. An SD card or external storage could help preserve data when connectivity is lost.
- Adding more sensors might require dynamic sensor configuration and more sophisticated scheduling for wake-ups. Current solution is tailored to a predefined sensors.

3.9 Code Structure

The project uses a standard ESP-IDF layout under the `main` folder, as shown below:

```
|- CMakeLists.txt
|- main
|   |- CMakeLists.txt
|   |- component.mk
|   |- gauge.c / gauge.h       (Battery gauge handling)
|   |- main.c / main.h         (Main application logic)
|   |- mqtt.c / mqtt.h         (MQTT client setup and event publishing)
|   |- rtc_wake_stub.c / .h    (Wake-up stub logic for low-power triggers)
|   |- sntp.c / sntp.h         (SNTP initialization and time synchronization)
|   |- wifi.c / wifi.h         (Wi-Fi provisioning, connection, event handling)
```

Below is a brief overview of the files:

- `main.c, main.h`: Contains application entry-point (`app_main`), handles device identification, event loops, power configuration, and sets up wake-up sources.
- `rtc_wake_stub.c, .h`: Implements the minimal RTC wake-up stub, including time checks, sensor event handling, and re-entry into deep sleep if needed.
- `wifi.c, wifi.h`: Initializes and manages the Wi-Fi connection (WPA2, SSID/password, IP assignment).
- `mqtt.c, mqtt.h`: Sets up MQTT connectivity, handles message publishing (e.g., PIR events, battery status).
- `sntp.c, sntp.h`: Configures SNTP for accurate time synchronization with NTP servers.
- `gauge.c, gauge.h`: Includes battery gauge drivers (`lc709203f`) to measure voltage and state-of-charge (RSOC).

3.10 Data Reporting (2p)

- Provide graphs proofing the continuous deployment of your sensors, including the battery usage history.
- Highlight and clarify interesting parts of the graphs.

4 Digital Twin (17 pages)

4.1 Event-driven Programming (1p)

- Explain the challenges of developing a distributed and event-driven system Focus on your knowledge (or lack of) about distributed systems, kubernetes, docker, event-driven programming, unfamiliarity with Python, new technologies, short-comings in time.
- What difficulties did you face? How did you approach debugging issues in the system? What tools did you use?

4.2 Our Digital Twin Basics (2p)

- Based on the communication diagram from the slides, describe which events are included in your code per component.
- Provide some reasoning about their frequency and purpose. What data is generated by your custom Event Fabric that is relevant to invoking remote resources? Explain your design decision.

4.3 Use case 1: Emergency Detection (2p)

- Given the uniqueness of your environment and your modeling approach, explain the design decision to determine whether a new data point might be normal behavior or an emergency.
- What metrics are used to compare the new data How does the detection evolve by re-training your model? How does the model evolve?
- Provide graphs comparing different weeks. Explain the behavior of the graphs. What is important to highlight? What features are relevant? Is the behavior expected?

4.4 Use case 2: Detection of behavioral changes (3p)

- Describe the modeling technique followed to build an understanding of stay duration in the rooms of your apartment.
- What does your model cannot capture?
- Explain the selection of your modeling approach.
- What is important to be aware of regarding your system and models?
- What metrics and values are considered to differentiate between an "information" and "To-do" message?
- How is the information communicated to the visualization component?
- How is the severity calculated?
- Is your modeling able to determine emergencies?

4.5 Use case 3: Behavioral change based on paths (3p)

Only present this if you did the path analysis.

- Give your path definition, the model structure, and the criteria for behavioral changes.
- Could you use path information also to report emergencies?

4.6 Use case 3: your own use case (4p)

Provide this section only, if you developed an own use case.

- If you implement another use case, explain the model, its use, important metrics, your modeling design decisions, and short-comings of the model given the amount of data, seasonality, style of data.

4.7 Performance Analysis (2p)

- Measure response times (time from trigger calling the event fabric until delivering the event to the scheduler or time from calling event fabric till actuation communicates an emergency) and execution times (time taken from dispatching the invocation until completing the invocation) for each event, break it down in terms of fetching data from storage and compute times.
- How long does it take to train your model?
- How does it perform when your model considers more data? Correlate increasing days of data vs time.
- What is the ratio between preprocessing your data and training?
- What is the size of your models?

4.8 Limitations (1p)

- Describe current limitations of your implementation.

4.9 Code structure (2p)

- Describe how your code is organized. Either individual folders per component or a unified folder with different "main.py" files.
- Did you face any challenges due to your unfamiliarity with the software and your code organization?
- Name the important files, their location, and their respective relevant implementation.

5 Declaration of data usage

Specify, whether you allow us to keep your data for tests with our own implementation. We make sure, that the data cannot be tracked down to your name.

5.1 Data Donation Agreement

Under this agreement, you understand that we, the Chair CAPS, can use (read, modify, delete) any sensor data generated during the WiSe2024 in future research and educational activities or produce derivative works. Given the current structure of the stored data in the procured Raspberry Pi, we ensure the data cannot reveal any personal information of the donor as it refers to a generic user: `iot-user`, while the fields contain generic names and values.

MTK, Signature

Date, Place

Figure 2: The caption explaining what can be seen in the image/figure. Readers often read captions first if they do not have much time. Thus, it is important to find a good short explanation.

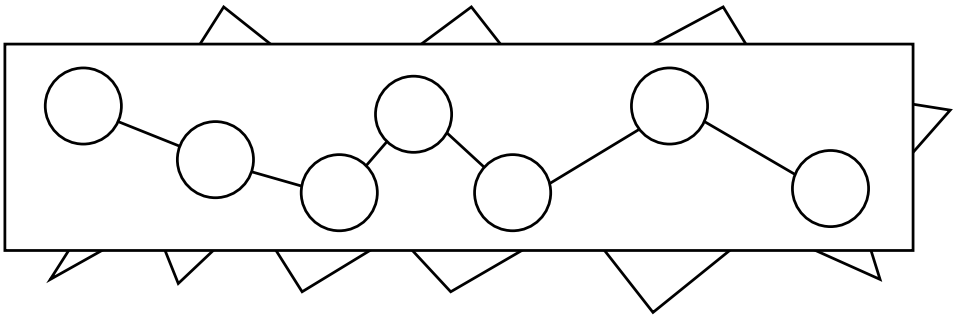

Figure 3: A nice caption. The larger width allows for more text without taking too much space.

The introduction of a scientific work usually consists of the following parts:

- motivation,
- issues or drawbacks with existing solutions of a problem at hand,
- overview of new contribution and rest of the paper.

In the motivation you should explain why a given topic is interesting at all, and also, why solutions are important from a scientific point of view. There already may be a lot of existing solutions. Thus, it is important for the reader to understand the issues with these solutions, and why they are not enough for e.g. a specific scenario.

After the motivation, you should explain the basic idea of your new contribution to solve the problem at hand, and give a short overview of how this works and why it is better than all other existing solutions. Finally, a short overview to the rest of the work should be provided, which may pick out the most important points. As any idea or solution proposed must be shown to be valid and useful, every scientific paper must have some evaluation and discussion. It may to useful to select important results, and mention them already as last part of the introduction, as motivation for the reader to read on. In summary, a good introduction makes the reader so interested into the topic and proposed new contributions that he cannot wait to read on.

6 Basic Rules for Using Latex

First, we want to refer to the figures and the introduction. See Figure **??** for the first floating figure with column width, and Figure **??** for the one using the full page width. And here, we want to put a reference to the introduction which is Section **??**.

In translating this template from German to English, I decided to stop here. There is not really much to get from the German text following. Anything Latex-related can also be looked up on the net. There is a *huge* number of tutorials, and so on.

Please do not use to much different font sizes and styles. It should be completely enough to go to *italic mode* for emphasizing something, such as newly introduced terms. You can refer to other parts of your paper (e.g. see Sec. **??**). Quoting in Latex is done "this way". Further, you may have problems with punctation characters. Most of them just need to be prefixed by a backslash, for others you may temporarily switch to math mode: $ & % # { } [ ] _ @ §< > \ @ ~/

Talking about math mode: you can do some very nice things this way:

$$a^2 + b^2 = c^2 \tag{1}$$

Again, refering to this equation is easy (see Eq. **??**). If you do not need numbering for equations, use the *displaymath* environment:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Short equations simply can be used within the regular text flow, such as with $x \to \infty$. Obviously, math is fun with Latex.

7 Enumerations

Enumerations using bullet points:

- this is the first item of this list of interesting facts,
- second item,
- and the last one.

They also can be numbered:

1. item one,
2. item two,
3. item three.

As shown, numbers always should be written out in the text, unless the belong to a title or a formula.

8 Literature

At the end of your paper, you should have a nice list of used literature. For scientific papers, this actually is needed. You always use other works as base for your own. Usually, you are

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Amount
Row 1	This column has a maximal width of 2 cm.	X	X	X	X	X	126,00
Row 2		This entry occupies three columns.			X	X	8,00
Sum							134,00

Table 1: This is the caption of the table.

not the only one thinking about a given difficult problem, so there is always related work which *must* be cited if known to the author.

Further, if you want to copy relevant sentences from an original paper, you *have* to cite them correctly, for example in this way:

> "I think there is a world market for maybe five computers." (T.J. Watson, IBM, 1943)

The rest of the work (especially all the regular text) must be written/phrased by you. If you write about some results or fact stated in another paper, you should refer to it. The 'Analytical Engine" — a mechanical calculation machine — created by Charles Babbage in the year 1838 was based on the decimal system [**?**].

9 Figures and Tables

No need to understand the following text.

Figures can span either one column (see Figure **??**) or the full page width (see Figure **??**). Latex automatically tries to find the best place for these floating figures. To influence that, you may move the figure a bit to the front of your text. As can be seen in Figure **??**, using images usually results in very bad quality. Better use vector formats: draw the figures with *xfig* or *inkscape*, and save them as PDF. As example of this procedure, see Figure **??**).

Similar to figures, tables can be referred to in the text (see Tab. **??**). However, sometimes it is useful to embed tables directly in the regular text flow:

	Column 1	Column 2
Row 1 Row 1		

10 Summary

The summary shortly repeats the core ideas and results from the previous text. If the reader has problems understanding the summary he knows that he should go back to the relevant sections. Thus, the last section should consist of:

- a summary,
- an evaluation of what was done, importance of this work,
- what is left, what still needs to be done,
- short outlook into the future.

Last but not least, we can explain anything missing yet in the evaluation done in this paper.
This allows to refer to what readers can expect from authors in the future.